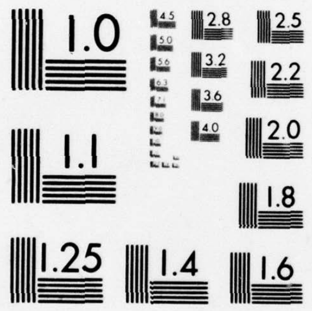
AD-A055 418 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL..ETC(U)
MAR 78 S J LARIMER
UNCLASSIFIED AFIT/GGC/EE/78-2 N/L

1 OF 3
AD
A055418


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AD A055418

AIR FORCE INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY
UNITED STATES AIR FORCE

SCHOOL OF ENGINEERING

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

78 5

AFIT/GGC/EE/78-2

(1)

AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL AND CONTINUOUS CONTROL SYSTEM ANALYSIS AND SYNTHESIS

THESIS

AFIT/GGC/EE/78-2 Stanley J. Larimer
2nd Lt USAF

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GGC/EE/78-2 ✓	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL AND CONTINUOUS CONTROL SYSTEM ANALYSIS AND SYNTHESIS		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stanley J. Larimer 2nd Lt, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Flight Dynamics Laboratory Wright-Patterson AFB, Ohio 45433		12. REPORT DATE March, 1978
		13. NUMBER OF PAGES 225
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

[illegible]

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer-Aided Design
Digital Control
Continuous Control

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report details the development of a computer-aided design program for continuous and discrete control systems. The program described is fully interactive and provides complete error detection, abort protection, and several levels of user assistance. In addition to digital and continuous time response, frequency response, and root locus, the program allows block diagram manipulation, state-space analysis, and a variety of continuous to discrete transformations. Built-in polynomial, matrix, and

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

scientific calculators are also provided. A user's manual and programmer's guide are included to aid in further development of the program.

AFIT/GGC/EE/78-2

AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL AND CONTINUOUS CONTROL SYSTEM ANALYSIS AND SYNTHESIS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Stanley J. Larimer, B.S.

Lieutenant USAF

Graduate Electrical Engineering

March 1978

1-B

072 225

Acknowledgements

I would like to gratefully acknowledge professor C. W. Richards, Capt T. E. Reeves, and Capt J. E. Colgate for their assistance in learning to use the FORTRAN IV computer language and the ASD computer center operating systems. Many thanks also go to Lieutenants Steven E. Enke, Eric R. Molner, and Richard E. Moore for the countless hours they spent testing the program and for their many helpful suggestions. Special appreciation is also expressed to Mr. Frank George and Mr. Brian Van Lied of the Air Force Flight Dynamics Laboratory for their help and sponsorship of the project.

I also wish to thank Dr. C. H. Houpis for his support and, of course, Dr. Gary B. Lamont whose never-ending suggestions doubled the amount of time spent on the project and tripled its quality.

Finally, I wish to thank my wife Pam for her tireless efforts in the preparation of this report.

Stanley J. Larimer

Contents

AFIT/GGC/EE/78-2

ABSTRACT

This report details the development of a computer-aided design program for continuous and discrete control systems. The program described is fully interactive and provides complete error detection, abort protection, and several levels of user assistance. In addition to digital and continuous time response, frequency response, and root locus, the program allows block diagram manipulation, state-space analysis, and a variety of continuous to digital transformations. Built-in polynomial, matrix, and scientific calculators are also provided. A user's manual and programmer's guide are included to aid in further development of the program.

AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL AND CONTINUOUS CONTROL SYSTEM ANALYSIS AND SYNTHESIS

I. Introduction

Guidance and control is a field characterized by problems which require a lot of computation. Many of these computations are conceptually simple, but require a lot of time and effort to perform. As a result, people working in the field spend a lot of time doing routine calculations instead of concentrating on problems worthy of their skill and training. The obvious solution is to use a computer to perform these tasks.

Background and Problem Statement

Many computer-aided design programs have been written with the intention of reducing the computational effort required to solve guidance and control problems. (Refs. 2, 3, 4, 5, 6, 7, 8, 9, 14, 22, 28, 29, 34, 35). Most perform the functions for which they were intended very well. Unfortunately many suffer from poorly designed user interfaces which, while not affecting their actual results, have seriously impaired their efficient use.

To point out this problem, some of the more frustrating experiences which inevitably occur when using computer-aided design programs are given on the next page. Not all programs suffer from each of these problems, but few avoid them all.

(1) The program terminates abruptly because the user accidentally typed an illegal character. All information typed is lost and the user must start over.

(2) In the middle of a long string of input data the user inadvertantly entered an incorrect value. The program runs its course producing meaningless output and when it terminates, the user must start over.

(3) The program terminates abruptly because an incorrect data value caused some internal equation to become computationally unsolvable. All current data is lost and the user must start over.

(4) The program actually operates correctly, producing the desired results and terminating normally. The user then wants to try another run with only one small change in the input. The user must retype all of the input data.

(5) The user finally obtains the needed results from a particular program. He then wishes to perform a different function on the same set of data. He starts another program and must retype his input data again, this time using a different format.

It is this continued typing and retyping of input data that makes current computer-aided programs difficult to use. If computer-aided design is to reach its full potential, some effort must be made to reduce this workload and to protect users from their own mistakes.

The problem is that computer-aided design programs are usually written only as they are needed to solve some particular set of problems currently of interest. Since the programmer is only writing the program as a quick means to some other end, he spends very little time in perfecting the program itself. He is generally satisfied, because of his own time constraints, if the program just accepts data and outputs correct answers. Few people have time to make programs operate conveniently.

As a result, there is a lot of computational assistance available today but little of it is easy to use. Savings in computation time which should be realized are lost in the effort of operating the programs themselves.

Statement of Purpose

If computer-aided design software is to be improved, it must be designed with two equally important goals in mind: (1) It must be capable of solving whatever class of problems is of interest, and (2) it should require as little effort from the user's standpoint as possible. Only if these goals are accomplished together will users be able to obtain all the benefits of computer-aided design.

The purpose of this investigation is to develop a computer-aided design program which will provide all of these benefits to people working in the field of guidance and control. To this end the following goals have been determined:

(1) To consolidate as many existing computer-aided design programs as possible into one package so that users can realize the benefits of a standard input format, the convenience of having every function at their fingertips, and the efficiency of a unified data structure where input does not have to be retyped for every phase of a problem.

(2) To create new program functions in areas where currently existing tools are unavailable or inadequate.

(3) To design and develop an efficient user interface which gives the user complete control of the program and its data with a minimum amount of typing.

(4) To develop a program structure which is easy to learn, maintain, modify, and extend.

Approach

The approach used in this investigation will be one of uniform, modular, standardized top-down design. While this approach is defined in detail in Chapter IV, the basic idea is a simple four step process:

(1) An overall picture is first obtained of what the program is to do and how it is to operate as a single large "black box." This is accomplished by essentially attempting to write the user's manual first so that every function the program is to perform is clearly defined.

(2) The single black box defined in step 1 is then broken down into several smaller boxes (modules) each of which must then be defined as thoroughly as the original box.

(3) Step 2 is repeated on each new set of boxes until eventually a level is reached where each module performs only a single elementary function.

(4) Finally, the elementary modules are constructed and tested individually using a standard programming style and then combined until eventually the original black box has been realized.

Thus, the program is designed "from the top down" to obtain a uniform final structure but built "from the bottom up."

Constraints

The following constraints were placed on this investigation either by the resources available or the nature of the problem itself:

(1) The goals must be sufficiently limited to allow accomplishment by a single individual in a 15 month period of time.

(2) The coding must be performed using the FORTRAN IV computer language because most of the currently existing programs which are to be included in the overall package have been written in this language. (Refs. 2,3,4,5,6,7,8,9,14,22,28,29,34,35) and it is generally accepted as a standard in the field. (Ref. 18)

(3) The program is constrained to operate in less than $60,000_8$ words of core memory by the computer system. (Ref. 1)

(4) The program must function reliably producing correct results.

Summary of Design Philosophy

Throughout the design, the following philosophy will be followed: "the user's time is more valuable than the compute's time." While it is true that a computer's time is more expensive on a minute for minute basis, it is much cheaper on a problem for problem basis (because of its speed). This is the only fair means of comparison since the object is to solve a certain number of problems at minimum expense, not to use up a certain amount of time at minimum expense! Therefore, the prime directive of this study will be: When a choice must be made, minimizing user time must take precedence over minimizing computer time. No effort will be spared to simplify the use of routines which are needed by so many people so much of the time.

II. Development of Initial Program Functions

As mentioned in the introduction, the objective of this design study is to develop a computer program which is both fully interactive and capable of solving the problems of interest. This chapter develops the requirements for the second of these goals. The first will be discussed in Chapter III.

The ultimate motivation behind this investigation is the desire to obtain a tool which is capable of performing every computation needed in the field of guidance and control. To demonstrate the magnitude of this undertaking, the following list has been compiled. While the list is by no means complete, it is representative of the kinds of functions which are needed. A discussion of *why* these functions are needed is beyond the scope of this report, but the fact that they *are* needed is documented (as indicated below) in more than one reference in the literature.

Needs for Computer Assistance in Guidance and Control

(1) Root locus in the s, z, w, and w' planes. (Refs. 13, 20, 21, 27, 28, 33)

(2) Discrete and continuous time response. (Refs. 14, 20, 21, 23, 27, 28, 30)

(3) Discrete and continuous frequency response. (Refs. 20, 21, 23, 27, 28, 30)

(4) Discrete, continuous, and stochastic system simulations. (Refs. 20, 23, 26, 30)

(5) Power spectrum analysis. (Refs. 25, 30)

(6) Sensitivity analysis. (Refs. 20, 21, 27, 28)

(7) Relative and global stability analysis. (Refs. 20, 21)

(8) Polynomial operations including addition, subtraction, multiplication, division, factorization, and expansion. (Refs. 13, 14, 20, 21, 23, 27, 28, 30)

(9) Matrix operations including:
Addition, subtraction, multiplication, inversion, and transposition.
Computation of adjoint, cofactor, and determinant.
Similarity transformations and rank determination.
Eigenvalue and eigenvector calculation.
State transition and resolvant matrix computation.
Diagonalization and Hermite normal form reduction.

(10) Discrete to and from continuous system representation transformations and approximations. (Refs. 14, 23, 25, 30)

(11) State-space model to transfer function model conversion. (Refs. 13, 14, 20, 21, 23, 25, 27, 28, 30, 34)

(12) Transfer function conversion to state-space canonical forms. (Refs. 14, 20, 21, 23, 27, 28, 30, 34)

(13) Conversion between state-space canonical forms. (Refs. 14, 20, 23, 27, 34)

(14) Continuous filter design including Butterworth, Chebyshev, elliptic, and other realizations. (Refs. 14, 23, 25, 30)

(15) Digital filter design including finite (FIR) and infinite (IIR) impulse response realizations. (Refs. 14, 23, 25, 30)

(16) Signal processing and analysis including convolution, Fourier transformation, integration, differentiation, correlation, and RMS analysis. (Refs. 14, 23, 25, 30)

(17) Classical control design including gain, lead, and lag compensation and Guillemin-Truxal techniques. (Refs. 20, 21, 27)

(18) Modern control including state-variable feedback, modal control, and observer design. (Refs. 20, 21, 27, 34)

(19) Direct digital design including minimal prototype, dead-beat response, windowing, frequency sampling and mean-square error techniques. (Refs. 14, 23, 25, 30)

(20) Optimal control design including solution of the Ricatti equation and performance index adjustment. (Refs. 20, 24, 26)

(21) Non-linear control design including phase plane methods, linearization, and curve fitting techniques. (Refs. 20 21, 22)

(22) Stochastic control analysis and Kalman filter design. (Refs. 21, 26)

(23) Partial fraction, continued fraction, and power series expansions. (Refs. 14, 20, 21, 22, 23, 27, 28)

(24) Equation solution including ordinary and partial differential equations and sets of linear algebraic equations. (Refs. 22, 28)

(25) Word length, quantization, and sampling rate analysis. (Refs. 14, 23, 30)

(26) Block diagram manipulation including addition and multiplication of transfer functions, and the closing of unity and non-unity feedback loops. (Refs. 13, 14, 20, 21, 23, 27)

Determination of Priorities

Because of the size of the above list, this study cannot hope to accomplish everything. Therefore, it is necessary to select which subset of these functions should be developed first. Such a selection should give priority to functions which are of greatest and most general use in the field, especially those which would be useful in performing other higher level functions. This section describes how priorities were assigned and what functions were selected for implementation during this investigation.

To aid in assigning priorities which would develop the most general and widely useful functions first, a survey was taken of textbooks in the field. (Refs. 13, 14, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 34). A priority was assigned to each function in relation to (1) the number of different areas of control (digital, continuous, stochastic, optimal, etc.) in which it was used and (2) the frequency with which it was needed in each area. No formal scoring procedure was used, but each function was evaluated according to the following general guidelines:

(1) Computations which were required extensively in more than one area were given top priority with those encountered most often being considered first. Not only are such functions the most generally useful, but they may often serve as building blocks for more complex functions.

(2) Specialized functions that were considered fundamental to a particular area of study were given the second highest priority.

(3) More advanced procedures needed for serious work in the field were selected next. At this point the functions became so specialized that priorities had to be assigned based on the needs of individuals already using the program as it was being developed.

(4) State-of-the-art computational techniques should be added to the complete package as they are developed.

Using these guidelines, the following areas were selected as being essential first level elements of a comprehensive computer program for guidance and control work:

(1) Root locus analysis.
(2) Discrete and continuous time response.
(3) Discrete and continuous frequency response.
(4) Block diagram manipulations.
(5) Polynomial operations.
(6) Matrix operations.
(7) Scalar operations.
(8) Classical control design techniques.
(9) Modern control design techniques.
(10) Continuous to discrete transformations.

These are the features that will be included in the program developed during this investigation. In addition to being a useful set of functions themselves, they are sufficiently general to allow development of a program structure which can accomodate any other function which may eventually be added. In other words, they provide a basis on which a data-base and an interactive user interface for the final program can be developed and tested. Development of the data base is the topic of the next section. The interactive user interface will be discussed in Chapter III.

Discussion of Data-base Variables Needed

In order to realize the goal of a unified expandable program where all routines use the same set of input data, it is necessary to develop a data-base of variables in which this information can be stored. While the actual realization of this data-base will be covered in Chapter IV, this section discusses what kinds of variables may be necessary to store all of the information needed by the program.

The most commonly needed piece of information for the functions to be realized is the transfer function. Root locus, time response, and frequency response routines all require input in this form as do many of the conventioanl control system design techniques. Block diagram manipualtions are, by definition, performed on transfer functions.

A transfer function is represented either as a ratio of polynomials or in factored form as a collection of poles, zeros, and gain constants. Thus, to represent a transfer function in the common data-base, provision must be made to store its numerator and denominator polynomial coefficients, the real and imaginary parts of its poles and zeros, and whatever gain constants may be associated with it.

The number of transfer function storage locations that should be provided is dependent upon the single function which uses the most of them, which in this case is the set of block diagram manipulations. Since these manipulations, in general, operate on two transfer functions to produce a third, at least three storage locations should be provided. An additional location for storing intermediate results would also be very useful, bringing the recommended minimum number of transfer functions to four. Naturally, if space permits, additional transfer functions could always be used to improve the versitility of the program.

Using similar reasoning, at least four arrays of polynomial coefficients should be provided to allow easy execution of polynomial operations. While such arrays could use the same locations in which transfer function numerators or denominators are stored, it is recommended (if space permits) that they be kept separate to minimize user confusion. Also, since corresponding to each polynomial there is a set of polynomial roots, it may be beneficial to provide storage locations for them. This would allow users to refer to polynomials in either factored or unfactored form.

Matrices form another large class of variables for which storage locations must be provided. At least five matrix arrays are needed to represent a system in state-space notation including: system ,input, output, direct transmission, and state variable feedback matrices If both discrete and continuous systems are to be represented simultaneously, an additional two arrays will be needed to store the discrete system and discrete input matrices These seven matrix locations will probably have to double as working registers for matrix arithmetic due to storage limitations.

Finally, each program in the package will undoubtably have its own set of scalar variables to use. Provision should be made to include such variables as they become necessary.

It must be remembered that, while the variables recommended above are sufficient for the needs of the routines developed in this study, it may be necessary to provide new storage locations as new features are added. For example, when signal processing routines are included, it may be necessary to add locations for storage of number sequences, and so on. Whatever data structure is developed should be flexible enough to allow continued expansion as the program grows.

III. Development of the Interactive User Interface

The second major goal of this investigation is to make a program which is not only computationally powerful, but also very easy to use. This requires that a user be able to specify, with a minimum amount of typing, exactly which operations the program is to perform and what data it is to use. The following list defines the features that a program must have if a truely efficient user interface is to be achieved:

(1) Protection against premature program termination due to input errors.

(2) Ability to recover from input errors without starting over.

(3) Ability to selectively display the value of any program variable at any time.

(4) Ability to selectively modify the value of any variable at any time.

(5) Ability to transfer the contents of any one variable to any other variable without manually retyping the data.

(6) Ability to use the output of one program function as the input to another without manually retyping the data.

(7) Ability to provide help to the user at any time, especially when input requested by the program is not understood.

(8) Ability to selectively list the options available to the user whenever needed.

(9) Ability to stop the program and restart it later without losing any data.

(10) Ability to abort a command without terminating the entire program.

(11) Ability to assume different modes of operation to tailor fit the program's performance to the user's preferences.

(12) Ability to control the flow of the program from function to function with complete freedom and as little effort as possible.

In short, an efficient user interface is one which requires a minimum amount of typing by the user while providing maximum control of the program.

Design of interactive features differs from the design of computational features in that the former must be built into the most basic parts of the program's internal structure while the latter can be added at anytime. Adding an interactive feature will, in general, require modifications to the entire structure from the ground up while adding another computational function generally has little effect on the rest of the program. For this reason, all of the user interface goals outlined above will be included as goals of this design study from the very beginning. It would be very difficult to add one at a later date.

For the purpose of discussion, an interactive user interface can be divided into four parts, including: program control, data-base control, error protection, and user assistance. The next four sections discuss these parts and what features they should include. While recommendations will be made concerning how these features might work, discussion of their actual implementation will be saved for Chapter IV.

Program Control Interface

Complete program control may be defined as the ability to randomly select program functions in any order at any time without regard for what functions may have been previously selected. Such control is highly desirable because it gives the user maximum flexibility to use the available functions to their fullest potential. It also protects the user from getting "locked in" to pre-selected sequences from which there is no escape short of complete program termination. Complete program control places all of the program functions at the user's finger tips and allows the user to define what operations are to be performed.

This kind of control, however, is not without its hazards. If the user is completely free to select functions in any order, he is also completely free to make mistakes. Great care must be taken when designing a control interface of this type, to ensure that when a user does make an error, it results only in an error message and not in a program abort.

This section discusses ways in which complete program control can be provided. The subject of error protection will be covered later in this chapter.

Two widely used methods for providing program control are to have the user type a command or select an option number. (Refs. 2, 3, 4, 5, 6, 7, 29) Each of these techniques has its own desirable features. Commands put the user in very direct control of the program and are particularly useful for simple, frequently used functions with few parameters. Unfortunately, when a program has a lot of options, commands require a user to learn a large vocabulary before he can use the program effectively. This is always undersirable, especially from a beginner's standpoint.

For the purposes of this program, it is recommended that the best features of both methods be used. When a function is simple and frequently used, it is easy to remember and may be provided as a command name. More complicated and less frequently used functions are more difficult to remember and may be best presented to the user as a list of option numbers to which he can refer.

One final item which should be discussed under this heading is the "switch" concept. Certain functions which allow the user to select his own favorite modes or operation (such as whether angles are calculated in radians or degrees, or whether the program should repeat (echo) all input it receives), may best be provided as a series of switches. Switches are simply logical variables which can be set either "on" or "off" by command from the user. Such switches may be used in any case where the user has a choice between one of two modes of operation. They are particularly beneficial in that the user can custom tailor the program to suit his own preferences. Very simply, a switch should be used for the type of command that is to remain in effect until changed by the user.

Data-base Control Interface

Complete control of the program functions alone is not enough to provide an efficient user interface. The user must also be provided with control of the data-base on which the functions operate. This is important because the user will often want to use data already stored in the computer with only minor changes in its form or location. Incomplete data-base control may require the user to do unnecessary retyping which is contrary to the goal of easy operation.

In this report, complete data-base control is defined as the ability to list, modify, and transfer the contents of any variable used by the program. This section discusses some of the possible techniques that can be used to give the user this kind of control.

The first element of a data-base control interface which must be developed is a reference system which will allow the user to designate which storage locations are to be operated upon. While some sort of reference system which assigns a number to each location in the data-base is a possibility, the preferred technique is to give every variable or variable array a name. The user can then refer to each storage location by typing its corresponding variable name. This system will work well as long as care is taken to see that each variable is given a logical, easy to remember name. A discussion of variable names chosen for the program being developed as a part of this study is given in Chapter IV.

To list the contents of any variable location, it is possible to have a command or option called "list variables" which, when selected, would ask the user to specify which variable(s) should be listed. A far more simple and direct approach, however, would be to have the user simply type the variable name as a command to list that variable. This technique is nearly optimal in the sense that it requires a minimum amount of typing by the user.

Similarly, there could be a command called "modify variables" which would ask the user to specify which variable to change and its new value. The easiest technique, however, is to simply type the variable's name and subscripts (if any) followed by an equal sign followed by its new value, all on one line as a command to make the indicated modification.

Transfer of information from one variable to another is a more complicated procedure. For simple scalar variables, the easiest technique is probably the same as for modification as described above. That is, the user can type the name of the variable to receive information followed by an equal sign followed by the name of the variable from which the information is to be obtained. For arrays of variables, where the meaning of an equal sign would be less clear, a special copy command could be used. One form of such a command could be: "COPY,VARA,VARB" which would instruct the program to copy the contents of the variable named VARA into the one named VARB. Either or both of these procedures would perform the required transfer operation with minimum user effort.

Error Protection and Recovery Interface

One of the most frustrating problems that plague interactive programs today is the devastating effect of inadvertant input errors. For the purpose of this discussion, these errors can be placed into three general classes:

(1) Typing an illegal character.

(2) Typing a number outside the range of acceptable values for a particular variable.

(3) Typing a number which will cause an arithmentic operation to become uncomputable.

All three error types can result in immediate termination of the program and complete loss of data. When this happens, the user is left with no recourse but to begin typing this input all over again.

Since most of the time spent using a program is devoted to typing in data, if the unnecessary retyping caused by these errors could be avoided, a tremendous improvement in efficiency could be achieved. This section discusses several ways in which the effects of inevitable user errors can be eliminated.

As long as standard FORTRAN READ routines are used to receive input from the user, there is little that can be done about common typographical errors. (Ref. 15:5-5). Once the READ routine has been called, control is out of the programmer's hands until the routine has successfully obtained the requested input. If an input error is encountered, the READ routine will either print the cryptic message: "ERROR, RETYPE RECORD AT THIS FIELD" or terminate the program completely. No opportunity for calling a recovery routine is ever provided.

The only good solution is to develop an alternate read subprogram to replace the standard FORTRAN one. Such a subprogram could read data in alphanumeric format (a standard

FORTRAN format which will accept <u>any</u> character) and then convert it to usable form using a translation routine. Any illegal characters detected when converting a line of data would be called to the user's attention by an understandable error message and the user given an immediate opportunity to make the needed correction. This approach would virtually eliminate the greatest single cause of abnormal program termination: the typographical error.

Errors caused by the user typing a number outside the range of acceptable values are easy to avoid. All that is necessary is to test each number when it is received to ensure that it is within its legal limits. If it is not, an error message can be printed and the user asked to retype the number.

Finally, errors due to illegal arithmetic operations must be eliminated. This is somewhat more difficult than the preceding kind of error because it requires the programmer to anticipate every possible arithmetic error. The only sure method is to test the argument of <u>every</u> operation that can produce such an error before the operation is executed. This can make programming somewhat more difficult but the resulting product is far more reliable. Table I lists some of the common functions which can cause abnormal program termination if they are given an illegal argument.

<u>User Assistance Interface</u>

To make an interactive program equally convenient for both the beginner and the experienced user, a special kind of

TABLE I

Some common functions that have possible illegal arguments.

Function	Symbolic Name	Error Condition
Division	A/B	$B = 0$
Natural Logrithm	ALOG(A)	$A \leq 0$
Common Logrithm	ALOG10(A)	$A \leq 0$
Square Root	SQRT(X)	$X < 0$
Arcsine	ASIN(A)	$\|A\| > 1$
Arccosine	ACOS(A)	$\|A\| > 1$
Exponential	EXP(A)	$\|A\| > 675.84$*

* Number is a function of word length and may differ from machine to machine.

user assistance interface must be developed. This interface must be able to teach a new user how to operate the program, provide quick memory refreshing and prompting for the familiar user, and yet stay completely out of the way of the expert who has no need for help.

The first and most important function that the user assistance interface must perform is to provide the user with a list of options that are available to him. The user should be able to obtain this list at any time but it should not be printed out automatically. Nothing is more irritating to an experienced user than to have to wait for a program to print unwanted information. Since a large program will, in general, have more options than a user will want to see at one time, the list of options should be divided into groups according to function so that the user can get a short list of options for the group in which he is interested.

Finally, provision should be made for several levels of prompting when the user is requested to input data. The experienced user may want his prompts to be short and to the point such as: "ENTER WMIN,WMAX > ". The new user, however, may not know that "WMIN" and "WMAX" are and will need a more detailed request for information. One possible solution to this need would be to have the program routinely give only brief prompts and if the user does not understand what is needed he can type a question mark for further details.

The thing that should be remembered is that users do not always have a user's manual handy when they are using a program. If the user assistance interface is properly designed, such a lack should not be a serious handicap.

Summary

An efficient user interface has been defined as one which gives the user maximum control of an interactive program with minimum input effort. The purpose of this chapter has been to develop the requirements for such an interface with the goal of making a program truely easy to use.

IV. Development and Realization of Internal Structure

The preceeding chapters have defined the needs and specified the goals for operation of the program to be developed. This chapter discusses the actual internal realization of the program. The external realization will be discussed with the results of this study in Chapter V.

The program to be described was given the name TOTAL to reflect the fact that it is intended to eventually perform the total range of computations needed in the field of guidance and control. The ten functional areas selected in Chapter II and all of the interactive features discussed in Chapter III were combined into a program with a form permitting continued growth. How this was accomplished is the subject of the following sections.

Design Approach

One of the problems with software design in the past has been that, in the interest of minimizing program size and memory requirements, program understandability and maintainability have been sacrificed. Programmers have relied on intuition, experience, and "pet tricks" to optimize their programs without realizing that such tricks make it very difficult for others to understand and use the coding. As a result, the life of most software has been very short because improving computer systems and changing needs soon render any one version of coding obsolete. If the original author of the program is no longer available, and if no one else can understand the coding well enough to bring it up-to-date, the program dies a rapid death. Software designers frequently find themselves "re-inventing the wheel" because it is easier to write an entirely new program than to understand and modify an old one. This is an obvious waste of time and resources.

Recently there has been an increasing awareness of the need for standardized programming techniques which are universally understandable. One such attempt to provide this standardization is the "Structured Analysis and Design Technique" (SADT) developed by SofTech, Inc. (Ref.31 :6). SADT is a highly developed methodology involving many functional analysis and system design concepts. Three of these concepts which were particularly useful in the development of TOTAL were, modularity, top-down design, and documentation. (Ref. 32, 2-1). The following paragraphs briefly discuss these concepts and how they were applied to TOTAL.

Modularity. Structured Analysis uses the concept of modularity to develop complex programs in a "divide and conquer approach." By successively breaking the program into more and more, smaller and smaller, well-defined modules, the analyst finally arrives at small enough pieces so that the function of each individual module can be easily understood and its interface to other modules clearly seen. Thus, the complex program that could not be understood in its total view can be well understood by seeing each of its modules and how they fit together. In fact, once this modular breakdown is created, replacement modules can be designed and "plugged-in" to improve the performance of the total program.

TOTAL was designed using this modular approach by making extensive use of subprograms and program overlays (to be discussed in this chapter). The most basic functions (such as polynomial multiplication, and elementary matrix row reduction) were developed first. These routines were then used as building blocks for higher-level functions (such as transfer function multiplication and matrix inversion) which in turn were used for still higher level functions (such as block diagram manipulation and state-space analysis). The modular design approach was put to such extensive use in the development of TOTAL that many of its programs and subprograms consist of almost nothing but calls to lower level subprograms.

Top-down design. In order to develop the modular decomposition described in the preceeding paragraphs, it was necessary to work "from the top down." The top-down design approach consists simply of viewing the program from the highest level, most general viewpoint, and then breaking down this view into finer and finer levels of detail. Without this approach, modular decomposition would be difficult.

For example, in the development of TOTAL, it was not known that polynomial multiplication and addition modules would be needed until the fundamental goal of block diagram manipulation capability was broken down into the need to add and multiply transfer functions which in turn required the routines to handle polynomials.

TOTAL was designed using the top-down approach as follows: First, the program was divided into ten functional areas as described in Chapter II. Each of these areas was then further divided into specific functions which were to become the individual options and other performance features in the finished program. Next, the necessary algorithms and procedures needed to perform each function were either located or developed. Finally, the specific blocks of coding and subprograms needed to implement each procedure were written. Throughout this top-down design process, decisions were made with program simplicity, size limitations, and interactive requirements in mind.

Documentation. The third fundamental concept used in the design approach for TOTAL was the need for continuous documentation. This concept is simply that documentation is best when it is produced continuously throughout the project while the design decisions are being made and can still be seen in context. (Ref. 31, 2-9) By recording why particular decisions were made and what factors influenced them, future extension and modification of the program can be made easier.

The primary reason for using this concept during the development of TOTAL, however, was that it helped to avoid a documentation phase at the end of the project. Since all steps taken in developing the program were recorded as they occurred, the resulting documentation precisely matched the final working program. Appendix B, the Programmer's Manual for TOTAL, is the result of attempts during this investigation to achieve this desired complete documentation.

Thus, the design process used in the development of TOTAL was modular, top-down, and documentation oriented. The remaining sections of this chapter discuss some of the more important design decisions that were made.

Overlays vs. Segmentation

To perform all of the functions specified as goals in the preceeding chapters requires a very large program. In fact, the programs and subroutines used in TOTAL collectively require more than $600{,}000_8$ words of central memory. Since many computer systems do not have this much memory available, (Ref. 1 :1) and since most limit interactive users to a much smaller amount (on the order of 60K) (Ref. 1 :52), it was necessary to design a program structure which would never require more than 60K at any one time. Two methods for fitting a large program into a small amount of memory are available including overlay generation and segmentation.

Overlay generation is simply a way of dividing a large program into a series of smaller programs each of which will fit into the available amount of memory. As each of these programs (overlays) is needed, it is loaded into memory replacing one which has just finished executing. A small executive routine, written to control the overall flow of the program, is responsible for calling each overlay into memory as it is needed. This executive is called the main overlay and remains in central memory at all times. The small programs which it controls are called primary overlays. Only the main overlay and one primary overlay are ever in central memory at a given time. Thus, the maximum amount of space needed by the entire program is just the sum of the space needed for the main overlay and the largest primary overlay.

Overlay generation is a simple process. It requires only the addition of a few new statements to the otherwise normal FORTRAN source code of the program.

Segmentation is a much more powerful method of subdividing a large program. Unlike overlays, it requires the addition of no new statements to the program. Instead, a separate set of control statements is written describing how the program is to be divided. A special routine called SEGLOAD then reads these control statements and divides a compiled version of the program, as directed, automatically. Once a segmented version of the program has been generated, it is used like any other program.

Because of the large number of sophisticated control directives that are available with SEGLOAD, segmentation is a versatile technique. For this reason, it is highly recommended by people who are thoroughly familiar with its use. Unfortunately, segmentation is very complex and therefore more difficult to learn. (A 41-page manual is needed to describe the process -- Ref. 17).

Thus, there is a trade-off between overlay generation and segmentation. Overlays are easier to use, but segmentation is more powerful. Such a choice might merit careful consideration if overlays were not capable of performing all of the functions needed. However, since overlays can do everything required in this case, the fact that segmentation is more powerful is irrelevant. In keeping with the goal of making the program easy to extend or modify for as many people as possible, the overlay technique must be selected. The more powerful features of segmentation are simply not needed.

Complete information on overlays and segmentation is given in Ref.17.

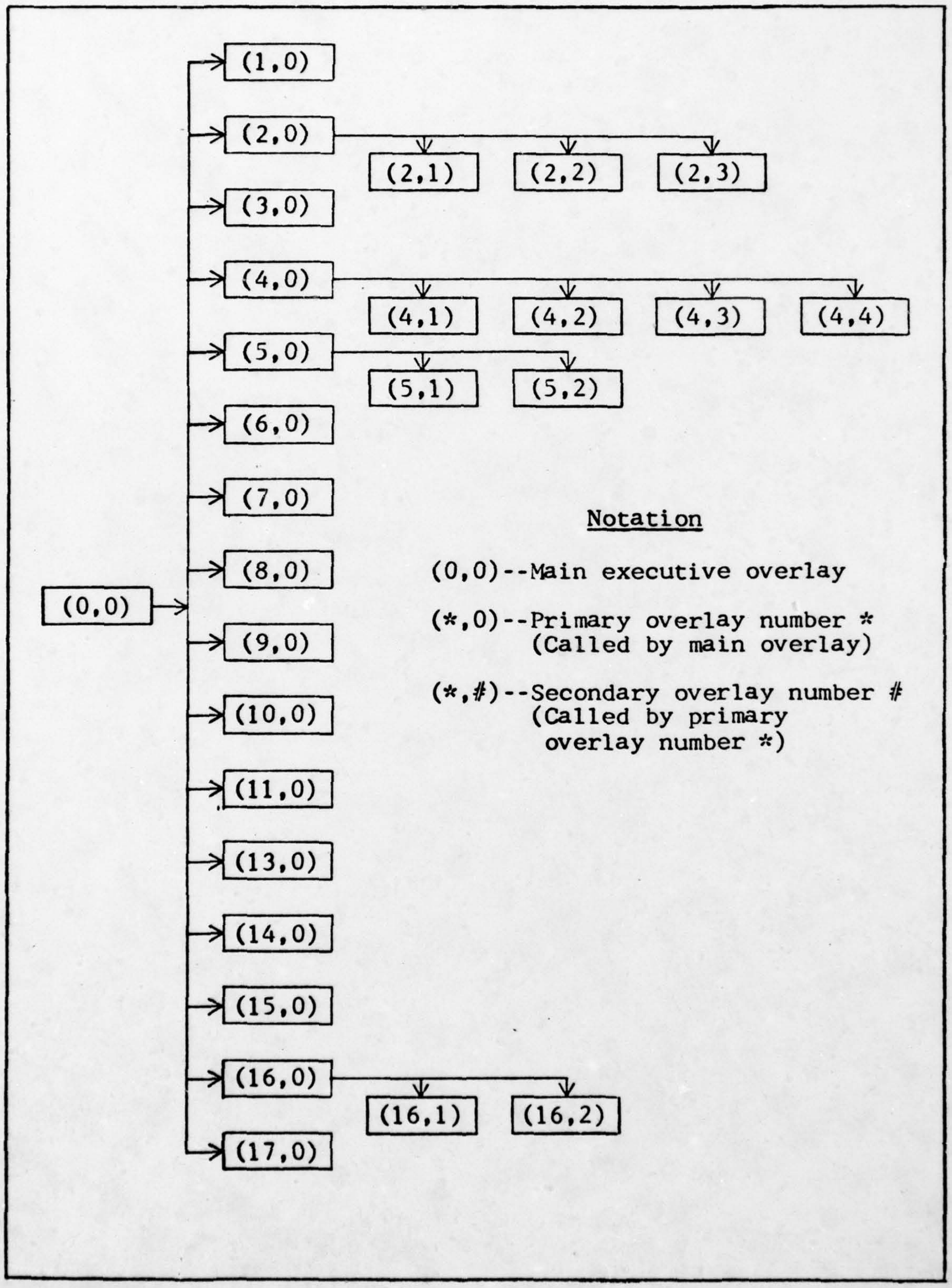
TOTAL's Overlay Structure

Once the overlay approach had been selected, it was only necessary to divide the program into approximately equal pieces small enough to fit in $60{,}000_8$ words of central memory. TOTAL was divided into one main overlay, seventeen primary overlays and eleven secondary overlays. A general flow chart showing this overlay structure is given in Fig. 1.

The main overlay (Overlay (0,0) in Fig. 1) holds the common data arrays, establishes their default values, and calls each primary overlay as it is needed. Since each program function may be performed, in general, by any one of these seventeen primary overlays, a short decision making routine is used to determine which overlay should be called. This routine is simply a massive computed GO TO statement in the main overlay called the "master overlay selector":

```
101    GO TO( 1, 3, 3, 3, 3, 3, 3, 3, 3, 9,
      +       14,14,14,14,14,14,14,14, 9, 9,
      +       17,17,17,17,14,14,14,17,17, 9,
      +        2, 2, 2, 2, 2, 2, 2, 2, 2, 9,
      +        4, 4, 4, 4, 4, 4, 4, 4, 4, 9,
      +        5, 5, 5, 5, 5, 5, 9, 9, 9, 9,
      +        3, 3, 3, 3, 3, 3, 3, 3, 3, 9,
      +       14,14,14,14,14,14,14,14,14, 9,
      +       16,16,16,16,16,16, 9, 9,16, 9,
      +        1, 9, 13, 9, 18, 9, 9, 9, 9, 9), NOPT
```

Entries in this GO TO statement are indexed by the option number, NOPT, and are simply the statement numbers of the seventeen overlay calling statements shown in Fig. 2. If a user selects, for example, option number 93, control is



<u>Notation</u>

(0,0)--Main executive overlay

(*,0)--Primary overlay number *
(Called by main overlay)

(*,#)--Secondary overlay number #
(Called by primary
overlay number *)

Fig. 1. TOTAL's overlay structure.

```
CALL OVERLAY UPDATE
1      CALL OVERLAY(5HTOTAL,1,0)
       GO TO 11111
CALL OVERLAY PARTL
2      CALL OVERLAY(5HTOTAL,2,0)
       GO TO 11111
CALL OVERLAY POLY
3      CALL OVERLAY(5HTOTAL,3,0)
       GO TO 11111
CALL OVERLAY ROOTL
4      CALL OVERLAY(5HTOTAL,4,0)
       IF(NOPT.EQ.48) GO TO 11
       GO TO 11111
CALL OVERLAY FREQR
5      CALL OVERLAY(5HTOTAL,5,0)
       GO TO 11111
CALL OVERLAY READER
6      CALL OVERLAY(5HTOTAL,6,0)
       GO TO 11111
CALL OVERLAY DECODER
7      CALL OVERLAY(5HTOTAL,7,0)
       GO TO 11111
CALL CALCULATOR (OVERLAY READER)
8      EXTCALC=.TRUE.
       CALL OVERLAY(5HTOTAL,6,0)
       GO TO 11111
CALL OVERLAY HELP
9      CALL OVERLAY(5HTOTAL,9,0)
       GO TO 11111
CALL OVERLAY DMULR
10     CALL OVERLAY(5HTOTAL,10,0)
       GO TO 11111
CALL OVERLAY TTYPLOT
11     CALL OVERLAY(5HTOTAL,11,0)
       GO TO 11111
CALL OVERLAY MISCELL
13     CALL OVERLAY(5HTOTAL,13,0)
       GO TO 11111
CALL OVERLAY MATRIX
14     CALL OVERLAY(5HTOTAL,14,0)
       GO TO 11111
CALL OVERLAY COPYIER
15     CALL OVERLAY(5HTOTAL,15,0)
       GO TO 11111
CALL OVERLAY XFORMS
16     CALL OVERLAY(5HTOTAL,16,0)
       GO TO 11111
CALL OVERLAY BLOCKER
17     CALL OVERLAY(5HTOTAL,17,0)
       GO TO 11111
```

Fig. 2. Primary overlay calling statements.

transferred to the statement in the main overlay whose number appears in the 93rd entry of the master overlay selector. In this case, statement number 13 would be selected which is just the calling statement for Overlay(13,0) (see Fig. 2). Overlay 13 then executes option 93 and returns control to the main overlay which repeats the entire process for the next user command. Complete details on the main overlay are given in Section 3 of Appendix B.

Primary overlays, like number 13, perform all of the actual operations in TOTAL. Each is responsible for a certain class of functions which may include option and command execution, variable definition and modification, switch setting, user assistance, and interactive user interfacing. If an overlay is too large for the given core restriction, it is divided into secondary overlays that will fit. Primary and secondary overlays are discussed in detail in Section 4 of Appendix B.

Data-base Development

An important aspect in the internal structure development of a program is the design of an effective data management system. Variables must be stored in a manner that will permit ready access by all parts of the program, easy modification and inspection by the user, and efficient use of computer memory. There are four possible techniques for storing information in a program. Each has its own advantages and disadvantages as described below:

(1) Local variables. Local variables are used within a single program or subroutine for storage of information while that particular routine is executing. When the routine has finished, the space in which such variables are stored is used for something else and the variable values are lost. Local variables are useful as scratch registers which are quickly accessable but which do not tie up any memory locations when not in use.

(2) Global variables. Often it is desired to keep certain variables in memory at all times so that they are available to any routine which needs them. Such variables are called global or common variables. They have an advantage in that they are quickly accessable and are not lost when execution passes from one routine to another. They have a disadvantage in that they take up memory locations at all times whether they are in use or not.

(3) Sequential-access files. In the event that large amounts of data must be stored, there may not be sufficient memory locations available in the computer. One possibility is to write all information to a local file on a disk or other mass storage device. If the data is to be used in a single block, the simplest technique is to use a sequential format. This means that data is written and read from the local file in the same order. This technique is slower than in-memory, but can handle much more information. It has an advantage over random-access storage in that it can be coded with ordinary FORTRAN WRITE statements. Its chief disadvantage is that information must be read from the file in the same order that it was written.

(4) Random-access files. Another form of mass storage which is similar to the sequential-access file is the random-access file. The chief difference and advantage is that information may be written and read on a random-access file in any order. The chief disadvantage is the more complicated input/output statements which are required.

All four types of storage are used in TOTAL's data structure to combine the advantages of each. Whenever possible, information needed only temporarily during the execution of a single routine is stored in local variables to conserve storage space. Information which is needed in more than one routine, or which is used repeatedly throughout the program is stored in global variables defined by labeled COMMON statements. A sequential access file is used to provide a backup memory of all global variables so that the program can be stopped and later restarted without loss of information. Finally, a random-access file is used as a mass storage device on which 24 additional transfer function arrays and 19 additional matrix arrays can be stored.

Selection of Variable Names

In the development of a data-base control interface as described in Chapter III, it was decided to assign a mnemonic reference name to every variable or variable array in the data-base. The only requirement placed on these names was that they should be assigned in a logical, easily remembered manner. This section discusses what names were chosen and how they were selected.

The data-base variables to be named were found to fall into five general categories including: transfer functions, polynomials, polynomial roots, matricies, and scalar constants.

Since these are natural divisions which are not difficult to remember, it was decided to use them as the basis for root words from which individual names could be formed. The following list shows the root words selected as reasonable mnemonic names for each of the five groups:

Group Type	Mnemonic Root Word
Transfer functions	TF
Polynomials	POLY
Polynomial roots	ROOT
Matrices	MAT
Scalar constants	K

Once a root word had been determined for each group, all that was necessary was to add one or two more letters to it to form a unique name for each variable in the group. These additional letters should naturally be selected in some logical manner as described below for each group:

Transfer functions. If transfer functions were to be divided into four types according to function, one possible partition might be: 1) forward transfer functions, 2) feedback transfer functions, 3) open-loop transfer functions, and 4) closed-loop transfer functions. This partition provides a convenient basis for naming the four transfer function arrays stored in the common data base. Since "G" and "H" are often used in the literature for the forward and feedback transfer functions respectively, (Ref.20, 21,27) and since "OL" and "CL" make reasonable abbreviations for "open-loop" and "closed-loop", the following name assignments were made for the four database transfer functions:

(1)	GTF	-- Forward transfer function
(2)	HTF	-- Feedback transfer function
(3)	OLTF	-- Open-loop transfer function
(4)	CLTF	-- Closed-loop transfer function

Polynomials. There are 12 polynomial arrays to be named. Four of the arrays are used as scratch registers for polynomial arithmetic and, for simplicity, were assigned the letters A, B, C, and D to distinguish between them. The remaining eight arrays are paired to form the numerator and denominator polynomials of the four transfer functions. Thus, it was logical to use the same letters (G, H, OL, and CL) in the names of each pair, the letters "N" for "numerator" and "D" for "denominator" were used. The resulting polynomial names are listed below:

(1)	POLYA	-- Coefficients of polynomial A
(2)	POLYB	-- Coefficients of polynomial B
(3)	POLYC	-- Coefficients of polynomial C
(4)	POLYD	-- Coefficients of polynomial D
(5)	GNPOLY	-- GTF numerator polynomial
(6)	GDPOLY	-- GTF denominator polynomial
(7)	HNPOLY	-- HTF numerator polynomial
(8)	HDPOLY	-- HTF denominator polynomial
(9)	OLNPOLY	-- OLTF numerator polynomial
(10)	OLDPOLY	-- OLTF denominator polynomial
(11)	CLNPOLY	-- CLTF numerator polynomial
(12)	CLDPOLY	-- CLTF denominator polynomial

Polynomial roots. Corresponding to the 12 polynomials named earlier are 12 arrays of polynomial roots. These arrays can be distinguished in a natural way using the same symbols used in the polynomial names and the root word "ROOT". Thus, ROOTA was used for the roots of POLYA, ROOTB for POLYB, and so on. However, because transfer function numerator and denominator roots are often called "zeros" and "poles" respectively, it was decided to modify the root word in these cases and substitute "ZERO" for "NROOT" and "POLE" for "DROOT". This scheme resulted in the following name assignments for the 12 root arrays:

(1)	ROOTA	--	Roots of POLYA
(2)	ROOTB	--	Roots of POLYB
(3)	ROOTC	--	Roots of POLYC
(4)	ROOTD	--	Roots of POLYD
(5)	GZERO	--	GTF ZEROS (roots of GNPOLY)
(6)	GPOLE	--	GTF POLES (roots of GDPOLY)
(7)	HZERO	--	HTF ZEROS (roots of HNPOLY)
(8)	HPOLE	--	HTF POLES (roots of HDPOLY)
(9)	OLZERO	--	OLTF ZEROS (roots of OLNPOLY)
(10)	OLPOLE	--	OLTF POLES (roots of OLDPOLY)
(11)	CLZERO	--	CLTF ZEROS (roots of CLNPOLY)
(12)	CLPOLE	--	CLTF POLES (roots of CLDPOLY)

Matrices. Since the letters A, B, C, D, F, G, and K are common matrix names used in digital and continuous control work, it was decided to use them for the seven matrix arrays in the data base. Along with the root word "MAT", these letters formed the names on the next page.

AMAT	--	Continuous system matrix
BMAT	--	Continuous input matrix
CMAT	--	Output matrix
DMAT	--	Direct transmission matrix
FMAT	--	Discrete system matrix
GMAT	--	Discrete input matrix
KMAT	--	State variable feedback matrix

Scalar constants. Twelve of the more than 60 scalar variables found in the data base are polynomial constants (highest order polynomial coefficients). These variables were designated with the root word "K" and the same distinguishing letters used for the polynomial arrays. The resulting variable names are listed below:

(1)	PAK	--	Polynomial A constant
(2)	PBK	--	Polynomial B constant
(3)	PCK	--	Polynomial C constant
(4)	PDK	--	Polynomial D constant
(5)	GNK	--	GTF numerator constant
(6)	GDK	--	GTF denominator constant
(7)	HNK	--	HTF numerator constant
(8)	HDK	--	HTF denominator constant
(9)	OLNK	--	OLTF numerator constant
(10)	OLDK	--	OLTF denominator constant
(11)	CLNK	--	CLTF numerator constant
(12)	CLDK	--	CLTF denominator constant

The remaining scalar variables were too numerous and different to be named in any standard manner. These variables were simply given names to reflect their individual functions. Fortunately, they are only used for special purposes throughout the program and can be defined as needed in appropriate sections of the user's manual (See Appendex A).

Realization of the Interactive User Interface

In Chapter III it was stated that an interactive user interface was needed to perform a variety of functions, including program control, data-base control, error protection and recovery, and user assistance. This section describes, in general terms, how each of these functions were realized.

Program control interface. This interface provides control to the user in two ways: option numbers and commands. Option numbers are used to allow the user to select any of the actual computation functions in the program. They were implemented by an input routine which, when it encounters a valid option number, executes a computed GO TO statement with a branch for each option. Commands were used to perform simple functions on the program itself such as the setting of a mode control switch or the transfer of information from one part of the program to another. They were implemented using another input routine which, when it encounters an alphanumeric name, consults a table of valid names and executes the command corresponding to the matching element. A complete development of the program control structure is given in Section 3 of the programmer's manual (see Appendix B).

Data-base control interface. The data-base control interface allows the user to (1) display the contents of a variable by typing its variable name, (2) modify the contents of a variable by setting its name equal to the desired new value, and (3) transfer the contents of one variable to another by equating variable names.

This interface was implemented as a translator routine which converts a string of user inputs such as variable names, equal signs, and numerical values into an array of coded numbers. Control is then provided by another routine which follows the array of coded commands and performs all of the necessary manipulations of variables in the data-base. The actual operation of this interface is discussed under programs READER and DECODER in Section 4 of the programmer's manual (see Appendix B).

Error protection and recovery interface. Error protection and recovery is provided automatically by all input routines used in the program. All input from the user is read first in alphanumeric format and then translated into useable form while being checked for errors. If an error is found, the user is notified, all correct information is printed up to where the error occurred, and the user allowed to continue typing from that point. Complete details on this interface are given under subroutine READS in Section 5 of the programmer's manual (see Appendix B).

User assistance interface. An entire program in TOTAL's overlay structure is devoted exclusively to printing user assistance messages. Whenever a need for help is noted by any of the other interfaces, control is transferred to this program and an appropriate message is printed. Further details on this interface are given in Section 4 of the programmer's manual.

Summary

This chapter has been written to provide an overview of how the internal structure of TOTAL was developed. It is intended to bridge the gap between the development of program specifications in Chapters II and III and the detailed accounts of the actual internal and external realizations presented in the appendicies.

V. Results and Recommendations

The goal of this investigation was to develop a computationally powerful, fully interactive computer program capable of growing into a comprehensive computer-aided design tool for the entire field of guidance and control. This chapter describes what was actually accomplished and where further work can be done.

Summary of Results

There were four major accomplishments produced as a result of this study. These accomplishments include a working computer program, a detailed user's manual, a fully documented programmer's manual, and this report. The following paragraphs discuss the nature and extent of each of these accomplishments.

The computer program. The most visible result of this study is the program TOTAL which is now in use by faculty and students at the Air Force Institute of Technology School of Engineering. TOTAL can best be described as "an interactive computer-aided design program for digital and continuous control system analysis and synthesis." This section describes, in general terms, the many functions which TOTAL can currently perform.

TOTAL is designed as a tool to be used with as much speed, agility, and confidence as one would use a familiar hand calculator. Its interactive interface allows the user complete freedom to select options, modify variables, and give commands without fear of abnormal program termination due to input errors of any kind. Its computational features provide an extensive array of powerful tools which may be used directly or combined to form even more powerful functions. The basic "building block" tools provided by TOTAL in its current state of development include:

(1) Discrete and continuous root locus analysis including tabular listings, printer plots, and Calcomp plots.

(2) Discrete and continuous time response analysis including tabular listings, printer plots, figures of merit computation, and Calcomp plots.

(3) Discrete and continuous frequency response analysis including tabular listings, printer plots, and Calcomp plots.

(4) Block diagram manipulations including addition and multiplication of transfer functions and closing of feedback loops.

(5) Discrete and continuous state-space to transfer function conversions.

(6) Continuous to discrete transfer function digitization using impulse invariance, bilinear, and first-difference transformations.

(7) Polynomial operations including addition, subtraction, multiplication, division, expansion, and factoring.

(8) Matrix operations including addition, subtraction, multiplication, inversion, transposition, matrix exponentiation, determinant computation, and Hermite normal form reduction.

(9) Calculation of state-transition and resolvant matrices.

(10) Scalar operations performed by a built-in 20-memory scientific calculator.

(11) Inverse Laplace and z-transforms.

(12) Partial fraction expansion of transfer functions.

The program also allows the user to define additional macro routines from the building blocks listed previously. Provision is made for these routines and all other information stored in the program to be saved in external storage files for use in subsequent runs of the program. Help is available at all times providing several levels of user assistance.

The features previously described form a very powerful subset of all the functions that are needed in the field. TOTAL has been designed with a structure that will allow easy addition of new capabilities as they are developed.

The user's manual. Development of a detailed user's manual was as important as the development of the program itself. Without such a manual, use of the program would have to rely on word of mouth instruction and many of the most sophisticated routines would remain forever unused. For this reason, great care was taken to ensure that every worthwhile bit of information concerning the program was included in a clear and logical manner. Liberal use of examples was made throughout the manual to further this end. The user's manual was produced as a separate document entitled User's Manual for TOTAL. It is included in this report as Appendix A.

The programmer's manual. The programmer's manual was written to provide documentation on all programs and subprograms used in TOTAL as well as a description of the overall structure and internal operation of the program. It is intended for the individual who wishes to make his own modifications and additions to the program and as a general aid in maintaining the program or transporting it from one computer system to another.

While the Programmer's Manual for TOTAL has been written as a separate document specializing on the internal structure of TOTAL, it is best used in conjunction with the user's manual and the rest of this report. This is because effective programming requires a knowledge of what a program is supposed to do and why it is supposed to do it in addition to how it is supposed to work. The programmer's manual is included in this report as Appendix B.

This report. This report has been written with the intent of explaining what was needed in computer-aided design for guidance and control, why it was needed, and the approaches taken to obtain it. Because separate documents (Appendices A and B) were written to explain the actual internal and external operation of the program, it was possible to omit this information from the body of the report. Such an approach streamlined the report considerably and allowed discussion of philosophical and developmental considerations without getting lost in eccessive detail. Together with the appendices, this report provides a thorough description of the program from its conception through its current state of development to its final realization.

Recommendations for Further Development

There are two directions in which further development computer-aided design software can be pursued: (1) the addition of interactive graphics capability and (2) the development of new program functions.

Interactive graphics is a technique in which the user controls the program with a light pen and in which output is displayed instantly in a variety of graphical forms.

The addition of interactive graphics features to TOTAL would be a project which, if successfully completed, would provide a quantum leap in available computer-aided design capability. Such a project would not require excessive rewriting of existing programs because TOTAL has already been developed with an interactive structure. The commands and option numbers which control TOTAL are ideally suited for display in a graphics menu format.

An area of greater interest to the guidance and control student is the development of additional program functions. All of the ground work and structure design for the program has been accomplished during this investigation so that future efforts need only be concerned with the theoretical and computational aspects of the new functions themselves. There is virtually unlimited room for continued development of computer-aided design tools in the field of guidance and control. The reader is referred to Chapter II for a list of some of the potential areas where further work is needed.

Bibliography

1. AFIT/EN. Digital Computer Manual for Faculty and Students of the School of Engineering (Third Edition). Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, July 1975.

2. AFIT/EN. FREQR. Frequency Response Program. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, 1974.

3. AFIT/EN. OPTCON. Solution of the Linear Quadradic Control Problem. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, 1974.

4. AFIT/EN. PARTL. Heaviside Partial Fraction Expansion and Time Response Program. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, 1974.

5. AFIT/EN. POLY. Polynomial Factoring Program. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, 1974.

6. AFIT/EN. ROOTL. User's Manual for a Digital Computer Routine to Calculate the Root Locus. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, August, 1974.

7. AFIT/EN. SVFB. State Variable Feedback Program. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, 1974.

8. AFIT/EN. ZTRAN. Z-Transform Program to Calculate the Discrete Time Response. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, 1974.

9. AFFDL-TR-74-69. Digital Flight Control Systems for Tactical Fighters. Honeywell, Inc. Wright-Patterson Air Force Base, Ohio: Air Force Flight Dynamics Laboratory, July 1974.

10. ASD Computer Center. ASD Computer Center Calcomp Plotter Guide (Revision A). Wright-Patterson Air Force Base, Ohio: ASD Computer Center, 1976.

11. ASD Computer Center. ASD Computer Center CDC NOS/BE User's Guide (Revision D). Wright-Patterson Air Force Base, Ohio: ASD Computer Center, 1976.

12. ASD Computer Center. ASD Computer Center Intercom Guide (Revision B). Wright-Patterson Air Force Base, Ohio: ASD Computer Center, 1976.

13. Blakelock, J. H. Automatic Control of Aircraft and Missiles. New York: John Wiley and Sons, Inc., 1975.

14. Cadzow, J. A. and H. R. Martens. Discrete-Time and Computer Control Systems. Englewood Cliffs, N. J.: Prentice Hall, Inc.

15. Control Data Corporation. FORTRAN Extended Version 4 Reference Manual (Revision B, Publication Number 60497800). Sunnyvale, California: Control Data Corporation, 1976.

16. Control Data Corporation. INTERCOM Version 4 Reference Manual (Revision B, Publication Number 60497600). Sunnyvale, California: Control Data Corporation, 1976.

17. Control Data Corporation. Loader Version 1 Reference Manual (Revision J, Publication Number 60342200). Sunnyvale, California: Control Data Corporation, 1976.

18. Control Data Corporation. SCOPE Reference Manual Version 3.4.4 (Revision J, Publication Number 60497600). Sunnyvale, California: Control Data Corporation, August, 1974.

19. Control Data Corporation. UPDATE Reference Manual. (Revision C, Publication Number 60342500). Sunnyvale, California: Control Data Corporation, 1976.

20. D'Azzo, J. J., and C. H. Houpis. Linear Control System Analysis and Design: Conventional and Modern. New York: McGraw-Hill Book Co., 1975.

21. DiStefano III, S. W. Feedback and Control Systems (Schaum's Outline Series). New York: McGraw-Hill Book Co., 1967.

22. Hornbeck, R. W. Numerical Methods. New York: Quantum Publishers, Inc., 1975.

23. Houpis, C. H., and G. B. Lamont. Short Course on Digital Systems/Information Processing (Second Edition). Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, July 1976.

24. Kirk, D. E., Optimal Control Theory, An Introduction. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1970.

25. McGillem, C. D., and G. R. Cooper. Continuous and Discrete Signal and System Analysis. New York: Holt, Rinehart, and Winston, Inc., 1974.

26. Meditch, J. S. Stochastic Optimal Linear Estimation and Control. New York: McGraw-Hill Book Co., 1969.

27. Melsa, J. L., and D. G. Schultz. Linear Control Systems. New York: McGraw-Hill Book Co., 1969.

28. Melsa, J. L., and S. K. Jones. Computer Programs for Computational Assistance in the Study of Linear Control Theory (Second Edition). New York: McGraw-Hill Book Co., 1973.

29. O'Brien, F. L. A Consolidated Computer Program for Control System Design. Unpublished master's thesis. Air Force Institute of Technology, December, 1976.

30. Oppenheim, A. V., and R. V. Schafer. Digital Signal Processing. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1975.

31. SofTech, Inc. An Introduction to SADT Structured Analysis and Design Technique. Waltham, Mass.: SofTech, Inc. November, 1976.

32. SofTech, Inc., Structured Analysis Reader Guide. Waltham, Mass.: SofTech, Inc., May, 1975.

33. Whitbeck, R. F., and L. G. Hofmann. Digital Control Law Synthesis in the w' Domain. Hawthorne, California: Systems Technology, Inc., August, 1977.

34. Wiberg, D. M. State Space and Linear Systems (Schaum's Outline Series). New York: McGraw-Hill Book Co., 1967.

35. Young; K. R. The Design of Automatic Control Systems Using Interactive Computer Graphics. Unpublished PhD dissertation. The University of Texas at Austin, May 1974.

APPENDIX A

USER'S MANUAL FOR TOTAL

(MARCH 1978)

Contents

OVERVIEW OF TOTAL

TOTAL is designed as a tool to be used with as much speed, agility, and confidence as one would use a familiar hand calculator. To help the casual user get a quick idea of what TOTAL is about, the following overview is provided.

* TOTAL is built around twelve general-purpose polynomials of maximum degree 50, and seven general-purpose 10 x 10 matrices.

* Eight of the polynomials may be paired to form the numerators and denominators of four general-purpose transfer functions.

* With just these polynomials, matrices, and transfer functions, the user is able to use the entire spectrum of TOTAL's capabilities, which include:

  - Add, subtract, multiply, divide, factor, expand, and copy polynomials.
  - Add, subtract, multiply, invert, transpose, obtain eigenvalues, preset, and copy matrices.
  - Obtain root locus, frequency response, and time response in both continuous and digital domain of open and closed-loop transfer functions.
  - Perform block diagram reduction.
  - Compute transfer functions from state-space.
  - Transform between continuous and digital domains using a variety of methods.

* A built-in 20-memory scientific calculator with a 4-register stack is available to the user at all times.

* The user may quickly list, transfer, or modify any variable, at any time, anywhere in the program.

* Complete error detection, diagnostics, and abnormal termination protection are provided. Specific help is available at any time by simply typing a question mark.

SECTION 1. INTRODUCTION TO TOTAL

The preceeding overview gave a capsule summary of what TOTAL can do. This section summarizes how it works.

TOTAL is an interactive program. The user has at his disposal 100 options and a few special commands with which he can manipulate a data base of 12 polynomials, 7 matrices, and roughly 60 scalar variables. Each option uses information stored in these variables to perform a particular function and saves the results for output or future use. During execution of TOTAL, the user has complete freedom to select options, modify variables, and give commands at will. The following paragraphs summarize what a person needs to know to utilize the full power of this program.

1.1 TOTAL'S INPUT MODES

TOTAL has three modes in which it pauses for the user to input information: OPTION, DATA, and CALCULATOR. Each mode has its own vocabulary of allowable inputs and its own characteristic prompt to the user.

<u>OPTION mode</u>.

OPTION mode is the primary command mode of TOTAL and is characterized by the prompt:

OPTION >

When TOTAL pauses in this mode, the user is free to type any command, select any option, modify any variable, or list any data. It is from this mode that the user controls the program.

As many commands, options, etc. may be typed on one line as desired. In fact, if the last item on a line is followed by a comma, TOTAL will wait for another line of input before beginning execution. Input items must be separated by commas or blanks.

DATA mode.

When an option has been selected and data is needed, TOTAL will ask for it. This is called DATA mode and is characterized by a prompt of the form:

ENTER ITEM1, ITEM2, ITEM3 >

In this mode, the user has a number of possible responses:

1. He can type in the desired values of the variables requested, separated by commas or blanks, in which case TOTAL will accept the data and continue what it was doing.

2. If he does not understand what "ITEM1, ITEM2, and ITEM3" are, he can type "?" and TOTAL will explain what information is needed.

3. If one of the numbers requested needs to be calculated, he can type "C" to enter CALCULATOR mode. When he returns from CALCULATOR mode, the prompt (and any numbers which he may have already typed) are repeated and TOTAL waits where he left off for the remaining input.

4. If one of the items is to have the same value as the last time it was requested (or a default value) the user does not need to input a number. He simply types a "*" in the place of the item to be left

unchanged and continues with the rest of the input.

5. If the user wants to know what the current values of the requested variables are, he can type "L" for a list. Numbers are listed in the order they were requested, the prompt (and any numbers typed before "L") are repeated, and TOTAL waits where the user left off for the remaining input.

6. If one of the numbers requested is currently stored in one of the calculator registers, the user may type X, Y, Z, T or R1 through R20 in place of the corresponding number. This is particularly useful when returning from CALCULATOR mode using the "C" option.

   In fact, the user can access any variable in TOTAL this way by typing "C" to go to the calculator, the variable's name to enter its value into the X register, "C" to return from calculator, and "X" to tell TOTAL to use the value now stored in the X register.

7. If, for any reason, the user wishes to abort and return to OPTION mode instead of continuing he may do so by typing "$".

The user may mix any of the above responses as needed while in DATA mode. TOTAL remains in this mode until all requested data has been supplied.

<u>CALCULATOR mode</u>.

CALCULATOR mode may be entered at any time by typing "C". The calculator operates like an HP-45 calculator (using reverse polish notation) and is designated with the prompt: **

When this prompt is displayed, the user may type a number, a key name (type KEYS for a list), or a "?" for a brief explanation of the calculator. CALCULATOR mode is terminated by typing another "C" whereupon TOTAL returns to the mode from which it came. Complete details on TOTAL's calculator are given in Section 5 of this manual.

1.2 TOTAL'S OPTIONS

TOTAL presently contains 100 options which have been divided into groups of 10 according to general function. The following is a list of the ten main groups.

0 - 9	Transfer function input options
10 - 19	Matrix input options
20 - 29	Block diagram manipulation and state-space options
30 - 39	Digital and continuous time response options
40 - 49	Root locus options
50 - 59	Digital and continuous frequency response options
60 - 69	Polynomial operations
70 - 79	Matrix operations
80 - 89	Digital and continuous transformation options
90 - 99	Miscellaneous options

The first option in each group simply lists the next ten options. For example, option 30 lists options 30 - 39. Complete details on these options are given in Section 2 of this manual. A tabulated list of all options is located inside the back cover.

1.3 TOTAL'S VARIABLES

Every variable in TOTAL's data base may be directly listed or modified by the user from OPTION mode. Typing a variable's name will list the current value of that variable. Typing a name, followed by an equal sign, followed by a number or another variable name will assign the value to the right of the equal sign to the variable on the left. Polynomial coefficients are referred to with subscripts, for example POLY(1) refers to the highest order coefficient in POLYA. Matrix elements are referred to with two subscripts.

A complete list of all variables is available at any time using option 97. Details on how to modify and use variables are given in Section 4.

1.4 HELP

Help is available to the user at all times and may be requested in two ways. In option mode, typing the command "HELP, option number" will give the user a short explanation of the option number specified. In all modes, the user may type "?" for assistance.

SECTION 2. COMPLETE DESCRIPTION OF OPTIONS

TOTAL presently contains 100 options arranged in groups of ten according to function. To execute an option, the user simply types the option number. TOTAL will then ask for needed information (if any) and execute the option.

The following is a detailed description of each option. It is intended primarily as a reference for the user who wants to know all the details about a particular option. The more casual user may wish to use the tabular option listing at the back of this manual for quick reference.

2.1 TRANSFER FUNCTION INPUT OPTIONS

Many of TOTAL's options center around the use or manipulation of transfer functions. Time response, frequency response, and root locus options, for example, all require the input of a transfer function prior to beginning calculations. Other options close feedback loops, add or multiply transfer functions, and perform related operations. TOTAL provides four working registers for this purpose:

GTF	Forward Transfer Function
HTF	Feedback Transfer Function
OLTF	Open-loop Transfer Function
CLTF	Closed-loop Transfer Function

All transfer function operations are performed with respect to one or more of these registers.

Each transfer function is made up of a numerator polynomial and a denominator polynomial which are stored in TOTAL as two arrays of polynomial coefficients. The transfer functions listed previously may be defined in terms of these polynomial arrays as:

$$\text{GTF} = \frac{\text{GNPOLY}}{\text{GDPOLY}} \qquad \text{OLTF} = \frac{\text{OLNPOLY}}{\text{OLDPOLY}}$$

$$\text{HTF} = \frac{\text{HNPOLY}}{\text{HDPOLY}} \qquad \text{CLTF} = \frac{\text{CLNPOLY}}{\text{CLDPOLY}}$$

Each of these polynomial array names has a specific meaning. For example, OLDPOLY stands for <u>OL</u>TF <u>D</u>enominator <u>POLY</u>nomial.

Since each polynomial can be factored into a constant times an array of roots, the four transfer functions may also be defined as:

$$\text{GTF} = \frac{\text{GNK} \cdot \text{GZERO}}{\text{GDK} \cdot \text{GPOLE}} \qquad \text{OLTF} = \frac{\text{OLNK} \cdot \text{OLZERO}}{\text{OLDK} \cdot \text{OLPOLE}}$$

$$\text{HTF} = \frac{\text{HNK} \cdot \text{HZERO}}{\text{HDK} \cdot \text{HPOLE}} \qquad \text{CLTF} = \frac{\text{CLNK} \cdot \text{CLZERO}}{\text{CLDK} \cdot \text{CLPOLE}}$$

where GZERO, GPOLE, HZERO, HPOLE, OLZERO, OLPOLE, CLZERO, and CLPOLE are complex arrays of polynomial roots and GNK, GDK, HNK, HDK, OLNK, OLDK, CLNK, and CLDK are corresponding constants. These variables are discussed in greater detail in Section 4.

Options 2 through 9 allow the user to input these transfer functions in either of the forms described above.

<u>OPTION 0</u>: <u>List options</u>.

This option gives a quick reference list of options 0 through 9. The user may use this, like all options ending in zero, to refresh his memory about the next ten options when this manual is not handy.

<u>OPTION 1</u>: <u>Recover all data from file MEMORY</u>.

While this is not a transfer function input option, it <u>is</u> an input option and is placed here because of its importance.

Option 1 reads all the values for every variable in TOTAL from a local file "MEMORY." During execution of TOTAL, the user may store the current values of all variables into memory by using option 91. (See Section 2.10) Data is automatically stored in MEMORY when the user types STOP.

Thus it is possible to end TOTAL, do something else, and restart the program later using option 1 to recover the data stored in MEMORY.

<u>OPTIONS 2, 3, 4, and 5</u>: <u>Polynomial form input</u>.

These options are identical in format and allow the user to input GTF, HTF, OLTF, and CLTF transfer functions, respectively, in polynomial form.

For example, if the user has an open-loop transfer function (OLTF) available as a ratio of two polynomials, he may input it using option 4, as shown on the following page.

Note that the input polynomials (OLNPOLY and OLDPOLY) were immediately factored and their roots stored in corresponding arrays (OLZERO and OLPOLE). The listing shown in boxes is suppressed if ECHO mode is off (see option 93, Section 2.10)

```
OPTION >  4

POLYNOMIAL INPUT OF OLTF(S)
ENTER NUM & DENOM DEGREES (OR SOURCE): 2 3

ENTER 3 NUMER COEFF--HI TO LO:
> 100 1000 1600

OLTF NUM COEFFICIENTS               OLTF ZEROS (OLZERO)
(    100.0     )S** 2      (     -2.000     ) + J(      0.          )
(    1000.     )S** 1      (     -8.000     ) + J(      0.          )
(    1600.     )          POLYNOMIAL CONSTANT=      100.0

ENTER 4 DENOM COEFF--HI TO LO:
> 1 26 154 680

OLTF DENOM COEFFICIENTS             OLTF POLES (OLPOLE)
(    1.000     )S** 3      (     -3.000     ) + J(      5.000       )
(    26.00     )S** 2      (     -3.000     ) + J(     -5.000       )
(    154.0     )S** 1      (     -20.00     ) + J(      0.          )
(    680.0     )          POLYNOMIAL CONSTANT=      1.000

GAIN= 1.0   OLK= GAIN*(OLNK/OLDK)= 100.

OPTION >
```

The user also has the option, instead of typing the actual numerator and/or denominator polynomial itself, of specifying another source of those numbers if they already exist in some other polynomial in TOTAL's memory. For example if the user responds, as underlined, to the prompt below:

```
ENTER NUM & DENOM DEGREES (OR SOURCE): 2,POLYA
```

it will still be necessary to enter the three numerator coefficients, but the denominator coefficients will be copied form POLYA automatically.

Similarly, typing:

```
ENTER NUM & DENOM DEGREES (OR SOURCE): CLNPOLY,POLYC
```

will tell TOTAL to get the numerator coefficients from CLNPOLY (Closed-Loop Numerator POLYnomial) and the denominator polynomial from POLYC. Thus, once information exists in any polynomial, it is available to any other polynomial.

The complete list of names the user may type includes: POLYA, POLYB, POLYC, POLYD, GNPOLY, GDPOLY, HNPOLY, HDPOLY, OLNPOLY, OLDPOLY, CLNPOLY, and CLDPOLY. For further information on these polynomials, see Sections 2.7 and 4.

OPTIONS 6, 7, 8, and 9: Factored form input.

These options are identical in format and allow the user to input GTF, HTF, OLTF and CLTF transfer functions, respectively, in factored form.

For example, if the user has a closed-loop transfer function (CLTF) available as a set of poles and zeros, he may input it using option 9, as shown on the next page.

Note that the roots were entered as actual x-y coordinates ($\sigma + j\omega$ values) in the s-plane. Also, note that the input roots (CLZERO and CLPOLE) were immediately expanded into polynomials and the coefficients stored in corresponding arrays (CLNPOLY and CLDPOLY). The listing shown in boxes is suppressed if ECHO mode is off (see option 93, Section 2.10)

```
OPTION >  9

FACTORED INPUT OF CLTF(S)
ENTER NUM & DENOM DEGREES (OR SOURCE): 2,3

ENTER NUMERATOR CONSTANT: 100

ENTER EACH ROOT--RE,IM
CLZERO( 1)= -2,0
CLZERO( 2)= -8,0

CLTF NUM COEFFICIENTS                CLTF ZEROS (CLZERO)
(   100.0     )S** 2      (     -2.000     ) + J(      0.        )
(   1000.     )S** 1      (     -8.000     ) + J(      0.        )
(   1600.     )          POLYNOMIAL CONSTANT=        100.0

ENTER DENOMINATOR CONSTANT: 1

ENTER EACH ROOT--RE,IM
CLPOLE( 1)= -3,5
CLPOLE( 2)= (     -3.000     ) + J(     -5.000     ) ASSUMED
CLPOLE( 3)= -20,0

CLTF DENOM COEFFICIENTS              CLTF POLES (CLPOLE)
(   1.000     )S** 3      (     -3.000     ) + J(      5.000     )
(   26.00     )S** 2      (     -3.000     ) + J(     -5.000     )
(   154.0     )S** 1      (     -20.00     ) + J(      0.        )
(   680.0     )          POLYNOMIAL CONSTANT=        1.000

CLK= (CLNK/CLDK)= 100.

OPTION >
```

The user also has the option, instead of typing the poles and/or zeros directly, of specifying another source of those numbers if they already exist in some other array in TOTAL's memory. (For examples, see explanation under options 2, 3, 4, and 5.) The complete list of names the user may type to specify a source include: ROOTA, ROOTB, ROOTC, ROOTD, GZERO, GPOLE, HZERO, HPOLE, OLZERO, OLPOLE, CLZERO, and CLPOLE. For further information on these root arrays see Section 4.

2.2 MATRIX INPUT OPTIONS

OPTION 10: List options.

This option gives a quick reference list of options 10 through 19.

OPTIONS 11 through 17: Matrix input options.

These options are identical in format and allow the user to input AMAT, BMAT, CMAT, DMAT, KMAT, FMAT, and GMAT, (as defined under option 19), respectively. For example, if the user wishes to input BMAT he would use option 12.

```
OPTION >  12

INPUT OF [ BMAT ] MATRIX:
ENTER MATRIX SIZE: ROWS,COLUMNS >  3,4

ENTER 4 ELEMENTS PER ROW:
ROW 1 > 11 12 13 14

ROW 2 > 21 22 23 24

ROW 3 > 31 32 33 34

  COL >   1            2            3            4
ROW
 1      11.00        12.00        13.00        14.00
 2      21.00        22.00        23.00        24.00
 3      31.00        32.00        33.00        34.00
```

The listing shown inside the inner box is suppressed if ECHO mode is off. (See option 93, Section 2.10)

Note that in the first example there were fewer rows than columns and the user was asked to enter BMAT by rows In the case where the opposite is true the user is asked to enter by columns.

```
OPTION >  16

INPUT OF [ FMAT ] MATRIX:
ENTER MATRIX SIZE: ROWS,COLUMNS >  5,2

ENTER 5 ELEMENTS PER COLUMN:
COLUMN 1 > 11 21 31 41 51

COLUMN 2 > 12 22 32 42 52

  COL >    1            2
ROW
 1       11.00        12.00
 2       21.00        22.00
 3       31.00        32.00
 4       41.00        42.00
 5       51.00        52.00
```

Once again the inner box is suppressed if ECHO mode is off.

OPTION 18: Help user set up state-space system model.

This option is a fast way to input all the matrices needed for a state-space model. It works in the same manner as inputting the matrices individually using options 11 to 17, but it dimensions each matrix automatically to make it conform with the given system.

Specifically, after stating whether the system is to be continuous (using AMAT, BMAT, CMAT, DMAT, and KMAT) or discrete (using FMAT, GMAT, CMAT, DMAT, and KMAT), the user is asked to type in the numbers of states (order of system), number of system inputs, and number of system outputs. This then determines the size of each matrix and the user is asked for the elements of each matrix in the same way as for options 11 to 17.

OPTION 19: Explain the use of the above matrices.

Option 19 simply prints the following message.

```
OPTION > 19
EACH MATRIX AVAILABLE TO THE USER IN OPTIONS 11-17 (TYPE
10 FOR LIST) MAY BE USED AS A SCRATCH/STORAGE REGISTER FOR
MANIPULATING MATRICES USING OPTIONS 71-79 (TYPE 70 FOR LIST).
IN ADDITION, THE MATRICES MAY BE USED TO REPRESENT DIGITAL
OR CONTINUOUS CONTROL SYSTEMS ACCORDING TO THE FOLLOWING
EQUATIONS:
CONTINUOUS:    XDOT(T) = AMAT · X(T)  +  BMAT · U(T)
                  Y(T) = CMAT · X(T)  +  DMAT · U(T)
      WHERE       U(T) = GAIN · (R(T) -  KMAT · X(T))
DISCRETE:       X(K+1) = FMAT · X(K)  +  GMAT · U(K)
                  Y(K) = CMAT · X(K)  +  DMAT · U(K)
                  U(K) = GAIN · (R(K) -  KAMT · X(K))
OPTION 18 IS AVAILABLE TO HELP THE NEW USER INPUT THE
REQUIRED MATRICES.  THE MORE EXPERIENCED USER MAY INPUT
THEM DIRECTLY USING OPTIONS 11-17.
OPTION >
```

The definitions for U(T) and U(K) above are standard state-variable feedback equations. If the feedback is not in state-variable form, KMAT may be set equal to zero and any feedback specified in transfer function form as HTF. (see options 3 and 6, Section 2.1)

2.3 BLOCK DIAGRAM MANIPULATION OPTIONS

OPTION 20: List options.

This option gives a quick reference list of options 20 through 29.

OPTION 21: Form OLTF from GTF and HTF.

This option multiplies GTF and HTF to form OLTF. The open-loop transfer function (OLTF) is defined in TOTAL as

$$\text{OLTF} = \text{GTF} \cdot \text{HTF}$$

$$= \frac{\text{GNPOLY}}{\text{GDPOLY}} \cdot \frac{\text{HNPOLY}}{\text{HDPOLY}}$$

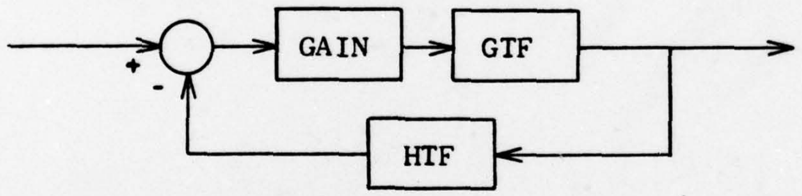
When option 21 is executed, the following operations are performed on variables in TOTAL.

$$\begin{aligned} \text{OLNPOLY} &= \text{GNPOLY} \cdot \text{HNPOLY} \\ \text{OLDPOLY} &= \text{GDPOLY} \cdot \text{HDPOLY} \\ \text{OLNK} &= \text{OLNPOLY}(1) \\ \text{OLDK} &= \text{OLDPOLY}(1) \\ \text{OLK} &= \text{GAIN} \cdot \text{OLNK}/\text{OLDK} \end{aligned}$$

In short, option 21 simply multiplies two transfer functions to form a new transfer function. This feature is thus useful for combining transfer functions in cascade.

OPTION 22: Form CLTF from GTF and HTF.

This option forms the closed-loop transfer function (CLTF) from a forward transfer function (GTF), a feedback transfer function (HTF), and a forward gain constant (GAIN) as follows:



$$\text{CLTF} = \frac{\text{GAIN} \cdot \text{GTF}}{1 + \text{GAIN} \cdot \text{GTF} \cdot \text{HTF}}$$

$$= \frac{\text{GAIN} \cdot \frac{\text{GNPOLY}}{\text{GDPOLY}}}{1 + \text{GAIN} \cdot \frac{\text{GNPOLY}}{\text{GDPOLY}} \cdot \frac{\text{HNPOLY}}{\text{HDPOLY}}}$$

$$= \frac{\text{GAIN} \cdot \text{GNPOLY} \cdot \text{HDPOLY}}{\text{GDPOLY} \cdot \text{HDPOLY} + \text{GAIN} \cdot \text{GNPOLY} \cdot \text{HNPOLY}}$$

where GAIN is some constant which the user can define.

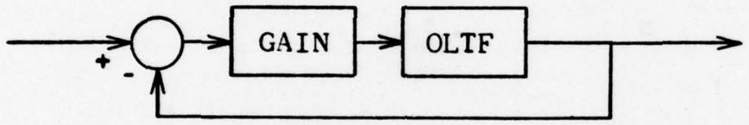
When option 22 is executed, the following operations are performed on variables in TOTAL.

$$\begin{aligned}
\text{CLNPOLY} &= \text{GAIN} \cdot \text{GNPOLY} \cdot \text{HDPOLY} \\
\text{CLDPOLY} &= \text{GDPOLY} \cdot \text{HDPOLY} + \text{GAIN} \cdot \text{GNPOLY} \cdot \text{HNPOLY} \\
\text{CLNK} &= \text{CLNPOLY(1)} \\
\text{CLDK} &= \text{CLDPOLY(1)} \\
\text{CLK} &= \text{CLNK/CLDK}
\end{aligned}$$

The user must supply GTF, HTF, and GAIN prior to selecting option 22. (See Sections 2.1 and 4.1)

<u>OPTION 23</u>: <u>Form CLTF from OLTF with unity feedback</u>.

If OLTF is a known open-loop transfer function for a system with unity feedback, this option will calculate the closed-loop transfer function CLTF as



$$\text{CLTF} = \frac{\text{GAIN} \cdot \text{OLTF}}{1 + \text{GAIN} \cdot \text{OLTF}}$$

$$= \frac{\text{GAIN} \cdot \dfrac{\text{OLNPOLY}}{\text{OLDPOLY}}}{1 + \text{GAIN} \cdot \dfrac{\text{OLNPOLY}}{\text{OLDPOLY}}}$$

$$= \frac{\text{GAIN} \cdot \text{OLNPOLY}}{\text{OLDPOLY} + \text{GAIN} \cdot \text{OLNPOLY}}$$

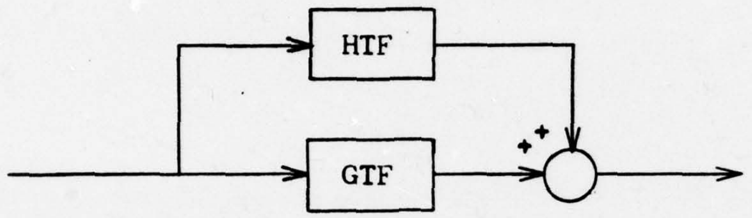
where GAIN is some numerical constant which the user can define.

When option 23 is executed, the following operations are performed on variables in TOTAL.

$$\begin{aligned}
\text{CLNPOLY} &= \text{GAIN} \cdot \text{OLNPOLY} \\
\text{CLDPOLY} &= \text{OLDPOLY} + \text{GAIN} \cdot \text{OLNPOLY} \\
\text{CLNK} &= \text{CLNPOLY(1)} \\
\text{CLDK} &= \text{CLDPOLY(1)} \\
\text{CLK} &= \text{CLNK/CLDK}
\end{aligned}$$

OPTION 24: Form CLTF from GTF + HTF (parallel)

If GTF and HTF are two forward transfer functions with the same input and outputs which are summed, this option forms a combined transfer function (CLTF) as follows:



$$\text{CLTF} = \text{GTF} + \text{HTF}$$

$$= \frac{\text{GNPOLY}}{\text{GDPOLY}} + \frac{\text{HNPOLY}}{\text{HDPOLY}}$$

$$= \frac{\text{GNPOLY} \cdot \text{HDPOLY} + \text{GDPOLY} \cdot \text{HNPOLY}}{\text{GDPOLY} \cdot \text{HDPOLY}}$$

When option 24 is executed, the following operations are performed on variables in TOTAL.

$$\begin{aligned} \text{CLNPOLY} &= \text{GNPOLY} \cdot \text{HDPOLY} + \text{GDPOLY} \cdot \text{HNPOLY} \\ \text{CLDPOLY} &= \text{GDPOLY} \cdot \text{HDPOLY} \\ \text{CLNK} &= \text{CLNPOLY}(1) \\ \text{CLDK} &= \text{CLDPOLY}(1) \\ \text{CLK} &= \text{CLNK/CLDK} \end{aligned}$$

The user must supply GTF and HTF prior to selecting option 25. (See Section 2.1)

OPTION 25: Form GTF(s) and HTF(s) from continuous state-space.

Given a state-space representation of the form

$$\begin{aligned} \dot{\underline{x}}(t) &= (\text{AMAT})\underline{x}(t) + (\text{BMAT})\underline{u}(t) \\ \underline{y}(t) &= (\text{CMAT})\underline{x}(t) + (\text{DMAT})\underline{u}(t) \end{aligned}$$

with state-variable feedback

$$\underline{u}(t) = \text{GAIN} \cdot (\underline{r}(t) - (\text{KMAT})\underline{x}(t))$$

option 25 computes the forward transfer function GTF between input i and output j as

$$\mathrm{GTF} = \underline{c}^{T}(sI - \mathrm{AMAT})^{-1}\underline{b} + d$$

and the equivalent feedback transfer function from output j to input i as

$$\mathrm{HTF} = \frac{\underline{k}^{T}(sI - \mathrm{AMAT})\underline{b}}{\underline{c}^{T}(sI - \mathrm{AMAT})\underline{b} + d}$$

where

$\underline{b}$ = i th column of BMAT

$\underline{c}^{T}$ = j th row of CMAT

d = ij th element of DMAT

$\underline{k}^{T}$ = i th row of KMAT

Using option 25 is a simple matter. The user first supplies five matrices using options 11, 12, 13, 14, and 15 (or option 18):

AMAT(NA,MA)	Continuous system matrix
BMAT(NB,MB)	Continuous input distribution matrix
CMAT(NC,MC)	Output matrix
DMAT(ND,MD)	Direct-transmission matrix
KMAT(NK,MK)	State variable feedback matrix

where

NA = MA = MC = MK =	Number of states
MB = MD = NK =	Number of inputs
NC = ND =	Number of outputs

(NOTE: For single input-single output (SISO) systems, BMAT becomes an NB by 1 column matrix, CMAT becomes a 1 by MC row matrix, DMAT becomes a scalar, and KMAT reduces to a 1 by MK row matrix.)

After the system matrices are set up, the user simply selects option 25. If the system is not SISO, TOTAL will ask which transfer functions are desired (between which input and which output) and then store the results in GTF and HTF.

When option 25 is executed, the following operations are performed on variables in TOTAL:

$$\begin{aligned}
\text{GNPOLY} &= (\text{CMAT}) \cdot \text{adj}(sI - \text{AMAT}) \cdot (\text{BMAT}) + (\text{DMAT}) \\
\text{GDPOLY} &= \det(sI - \text{AMAT}) \\
\text{HNPOLY} &= (\text{KMAT}) \cdot \text{adj}(sI - \text{AMAT}) \cdot (\text{BMAT}) \\
\text{HDPOLY} &= \text{GNPOLY}
\end{aligned}$$

$$\begin{aligned}
\text{GNK} &= \text{GNPOLY}(1) & \text{HNK} &= \text{HNPOLY}(1) \\
\text{GDK} &= \text{GDPOLY}(1) & \text{HDK} &= \text{HDPOLY}(1) \\
\text{GK} &= \text{GNK}/\text{GDK} & \text{HK} &= \text{HNK}/\text{HDK}
\end{aligned}$$

and, as always, the roots of GNPOLY, GDPOLY, HNPOLY, and HDPOLY are automatically stored in GZERO, GPOLE, HZERO and HPOLE respectively.

<u>OPTION 26</u>: <u>Form GTF(z) and HTF(z) from discrete state-space</u>.

Option 26 functions exactly like 25 except that FMAT and GMAT are used in place of AMAT and BMAT. (If the same data are used, identical transfer functions result.) Option 26 is intended to reduce a discrete model of the following form to transfer functions in z-plane.

$$\begin{aligned}
\underline{x}(k+1) &= (\text{FMAT})\underline{x}(k) + (\text{GMAT})\underline{u}(k) \\
\underline{y}(k) &= (\text{CMAT})\underline{x}(k) + (\text{DMAT})\underline{u}(k) \\
u(k) &= \text{GAIN} \cdot (\underline{r}(k) - (\text{KMAT})\underline{x}(k))
\end{aligned}$$

<u>OPTION 27</u>: <u>Write adjoint of (sI - AMAT) to file ANSWER</u>.

The adjoint of (sI - AMAT) is a matrix of polynomials in s (or z) which is useful in computing the resolvant matrix:

$$(sI - \text{AMAT})^{-1} = \frac{\text{adj}(sI - \text{AMAT})}{\det(sI - \text{AMAT})}$$

Option 27 tabulates each element of the adjoint matrix (adj(sI - AMAT)) as a column of polynomial coefficients listed from highest to lowest power and writes them to local file

ANSWER for later disposition to a line printer.

For example, given the matrix

$$\text{AMAT} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & -2 & 0 \\ -2 & 0 & -3 \end{bmatrix}$$

the corresponding adjoint matrix is

$$\text{adj}(sI - \text{AMAT}) = \begin{bmatrix} s^2 + 5s + 6 & 0 & 0 \\ s + 3 & s^2 + 4s + 3 & 0 \\ -2s - 4 & 0 & s^2 + 3s + 2 \end{bmatrix}$$

and option 27 tabulates it as follows:

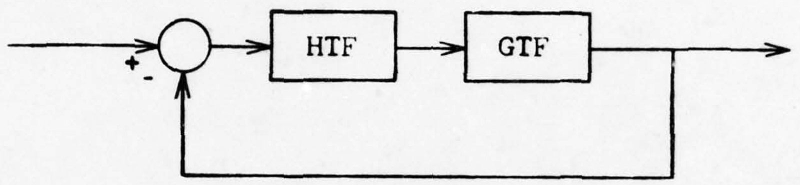
```
[                                          ]
|  1.00    |    0.     |      0.           |        S** 2
|  5.00    |    0.     |      0.           |        S** 1
|  6.00    |    0.     |      0.           |        S** 0
|          |           |                   |
| ---------+-----------+------------------ |
|          |           |                   |
|  0.      |    1.00   |      0.           |        S** 2
|  1.00    |    4.00   |      0.           |        S** 1
|  3.00    |    3.00   |      0.           |        S** 0
|          |           |                   |
| ---------+-----------+------------------ |
|          |           |                   |
|  0.      |    0.     |      1.00         |        S** 2
| -2.00    |    0.     |      3.00         |        S** 1
| -4.00    |    0.     |      2.00         |        S** 0
[                                          ]
```

The numerator and denominator of each element in the resolvant matrix can thus be obtained using option 27 (to find adj(sI - AMAT) and option 71 (to find det(sI - AMAT), respectively.

<u>OPTION 28</u>: <u>Guillemin-Truxal cascade compensator design</u>.

The Guillemin-Truxal cascade compensator design technique involves selecting a desired closed-loop transfer function

(CLTF) for a known plant (GTF) and then finding a cascade compensator (HTF) that, with unity feedback, will give the desired CLTF. (See D'Azzo and Houpis, Linear Control System Analysis and Design. pp 408-410).



Option 28 simply solves for the HTF that will give the desired results as follows:

$$\text{CLTF} = \frac{\text{GTF} * \text{HTF}}{1 + \text{GTF} * \text{HTF}}$$

or

$$\text{CLNPOLY} = \frac{\frac{\text{GNPOLY}}{\text{GDPOLY}} * \text{HTF}}{1 + \frac{\text{GNPOLY}}{\text{GDPOLY}} * \text{HTF}}$$

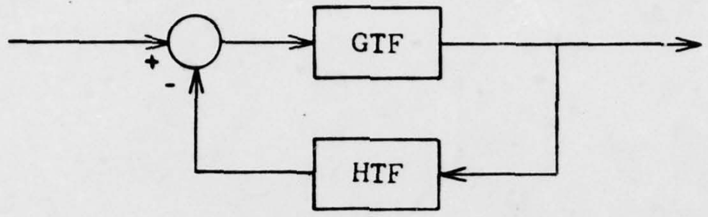
or, solving for HTF:

$$\text{HTF} = \frac{\text{HNPOLY}}{\text{HDPOLY}} = \frac{\text{GDPOLY} * \text{CLNPOLY}}{\text{GNPOLY} * (\text{CLDPOLY} - \text{CLNPOLY})}$$

The user must supply GTF and CLTF prior to selecting option 28. (See Section 2.1)

OPTION 29: Guillemin-Truxal feedback compensator design.

The Guillemin-Truxal feedback compensator design technique involves selecting a desired closed-loop transfer function (CLTF) for a known plant (GTF) and then finding a feedback compensator (HTF) that will give the desired CLTF.



Option 29 simply solves for the HTF that will give the desired results as follows:

$$\text{CLTF} = \frac{\text{GTF}}{1 + \text{GTF} * \text{HTF}}$$

or

$$\frac{\text{CLNPOLY}}{\text{CLDPOLY}} = \frac{\frac{\text{GNPOLY}}{\text{GDPOLY}}}{1 + \frac{\text{GNPOLY}}{\text{GDPOLY}} * \text{HTF}}$$

or, solving for HTF,

$$\text{HTF} = \frac{\text{HNPOLY}}{\text{HDPOLY}} = \frac{\text{GNPOLY} * \text{CLDPOLY} - \text{GDPOLY} * \text{CLNPOLY}}{\text{GNPOLY} * \text{CLNPOLY}}$$

The user must supply GTF and CLTF prior to selecting option 29. (See Section 2.1)

2.4 TIME RESPONSE OPTIONS

The next ten options (30-39) perform continuous or discrete time response analysis of either open or closed-loop transfer functions. To make fullest use of these options, the user should be aware of three important mode control switches: CLOSED, ANSWER, and TSAMP. These switches are described in the following paragraphs.

CLOSED is a switch which determines whether the response calculated will be that of the open-loop transfer function (OLTF) or the closed-loop transfer function (CLTF). Typing "CLOSED, ON" selectes CLTF while "CLOSED, OFF" selects OLTF. Similarly, typing "ANSWER, ON" will cause all output to be written to a local file ANSWER while "ANSWER, OFF" displays the output at the user's terminal. For further information on these switches, see option 93. (Section 2.10)

TSAMP is not truely a switch in the on-off sense. It is a variable containing the value of the sampling time in seconds. When TSAMP = 0, TOTAL assumes that the system to be analyzed is continuous and expressed in terms of the Laplace operator s. <u>If TSAMP is given some positive value, the system is considered to be discrete in terms of the z-transform operator z with the specified sampling time</u>.

Note that all switches must be set while still in OPTION mode <u>before</u> typing the desired option number. For example, typing: OPTION > <u>ANSWER</u>, <u>ON</u> <u>CLOSED</u>, <u>OFF</u> <u>TSAMP</u> = <u>.05</u> <u>31</u> will execute option 31 with the switches set as shown. Switches always remain as set until changed by the user and need not be retyped before every option.

Prior to selecting any option, the user must supply a transfer function using an appropriate input option (options 0-9, Section 2.1) Functions are restricted to no more than one repeated root which must be real. For the continuous case, the number of zeros must be less than the total number of poles and not more than one greater than the number of non-repeated poles. For discrete responses, the order of the numerator must not exceed that of the denominator.

OPTION 30: List options.

This option gives a quick reference list of options 30 through 39.

OPTION 31: Tabular listing of F(T) or F(K).

This option provides a tabular listing of the time response values over a specified range of time. For continuous responses (TSAMP = 0), the user is asked to specify an initial time, final time, and the desired time increment, and the values of T and F(T) are tabulated. For discrete responses (TSAMP $\neq$ 0), where CLTF or OLTF has been supplied as a z-transfer function, the user is asked to enter the initial K, final K, and K increment for F(K) where K is an index corresponding to a time T = K * TSAMP. In this case K, F(K), input R(K), and K * TSAMP are tabulated as shown below for a pulse input and

$$\mathrm{CLTF}(z) = \frac{6.0z}{(z + 0.2)(z - 0.7 \pm j0.2)}$$

The input R(K) for either continuous or discrete systems may be selected using option 39.

```
OPTION >  31

   DISCRETE TIME RESPONSE FOR CLTF(Z)
   WITH PULSE INPUT OF STRENGTH = 1. AND WIDTH = 5. SAMPLES

ENTER INITIAL K, FINAL K, & K INCREMENT FOR F(K) >  0 10 1


       K          F(K)            INPUT R(K)           K*TSAMP

       0          0.                1.000             0.
       1          0.                1.000             .200000
       2          6.000             1.000             .400000
       3          13.20             1.000             .600000
       4          20.34             1.000             .800000
       5          26.47             1.000             1.00000
       6          31.28             0.                1.20000
       7          34.76             0.                1.40000
       8          31.09             0.                1.60000
       9          25.30             0.                1.80000
      10          18.90             0.                2.00000
```

OPTION 32: Plot F(T) or F(K) at user's terminal.

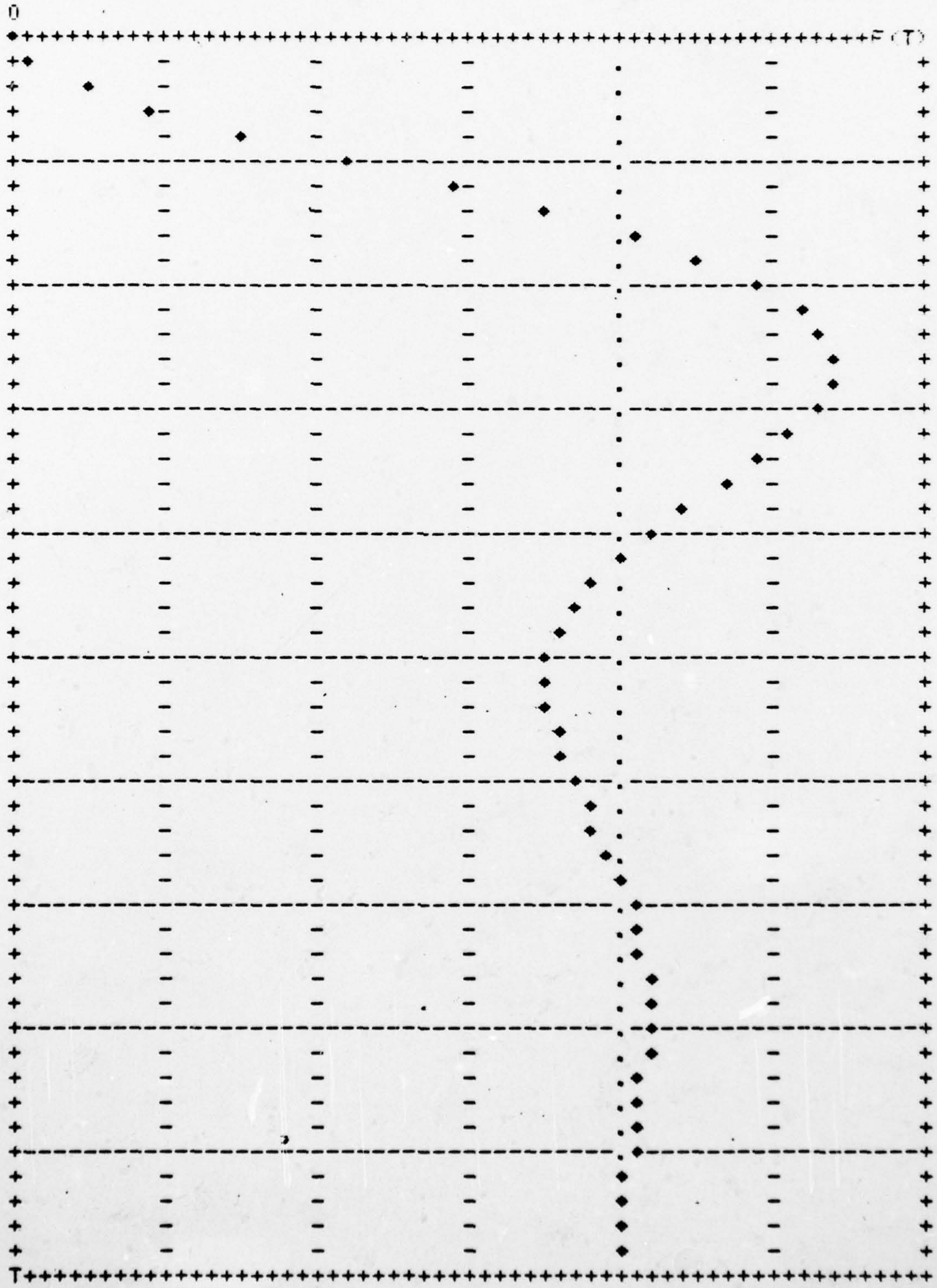
This option prints out a 6 by 8 inch plot of F(T) over an interval of time specified by the user. The F(T) and T axes are scaled identically to those of the Calcomp plot in option 34 and thus this option may be used to preview (within limits of printer resolution) exactly how a Calcomp plot using option 34 would appear.

For discrete systems (TSAMP $\neq$ 0), F(K) is plotted. In this case, the plot will not be 8 inches long. Instead, the increment of K that is plotted will be selected so that the plot length will never exceed 8 inches.

F(T) and F(K) are normally scaled automatically, however, the user may select his own scale by typing SCALE, OFF prior to selecting the option.

```
ENTER INITIAL TIME, FINAL TIME > 0 2

REGION OF CALCULATION:  T=      0.          TO     T=      2.000
                     F(T)=      0.          TO F(T)=       1.500
```



```
GRID SCALE:  T-AXIS:      .2000      SECONDS/DIVISION
            F(T) AXIS:     .2500       UNITS/DIVISION
```

A sample plot is shown on the preceeding page.

OPTION 33: Printer plot.

This option is identical to option 32 except that the plot generated is twice as wide, twice as long, and intended exclusively for disposition to a line printer. The plot is always written to a local file called ANSWER (regardless of whether the switch ANSWER is on or off). After execution of TOTAL has been terminated, the user may send the file ANSWER to the printer using the ROUTE or DISPOSE commands.

OPTION 34: Calcomp plot of F(T) or F(K).

This option produces a Calcomp plot and stores it in a local file PLOT which the user can later dispose to the plotter when TOTAL has been terminated.

The user is first asked to enter the initial and final time to be plotted. Since the time axis will have ten divisions on the plot, it is usually wise to select the duration of the plot to be a number evenly divisible by ten. If the switch SCALE is on, TOTAL will automatically scale the magnitude axis to even divisions which will extend over the range of the function in the region of time to be plotted. If SCALE is off, the user will be asked to specify the minimum and maximum axis values. In this case, since the F(T) axis is always six divisions long, the axis length should be chosen to be evenly divisible by six for best results.

Finally the user is asked to input a title to be drawn above the plot. The title may have up to 50 characters.

A guide which is 50 characters long (including the brackets) is printed to aid the user as shown below:

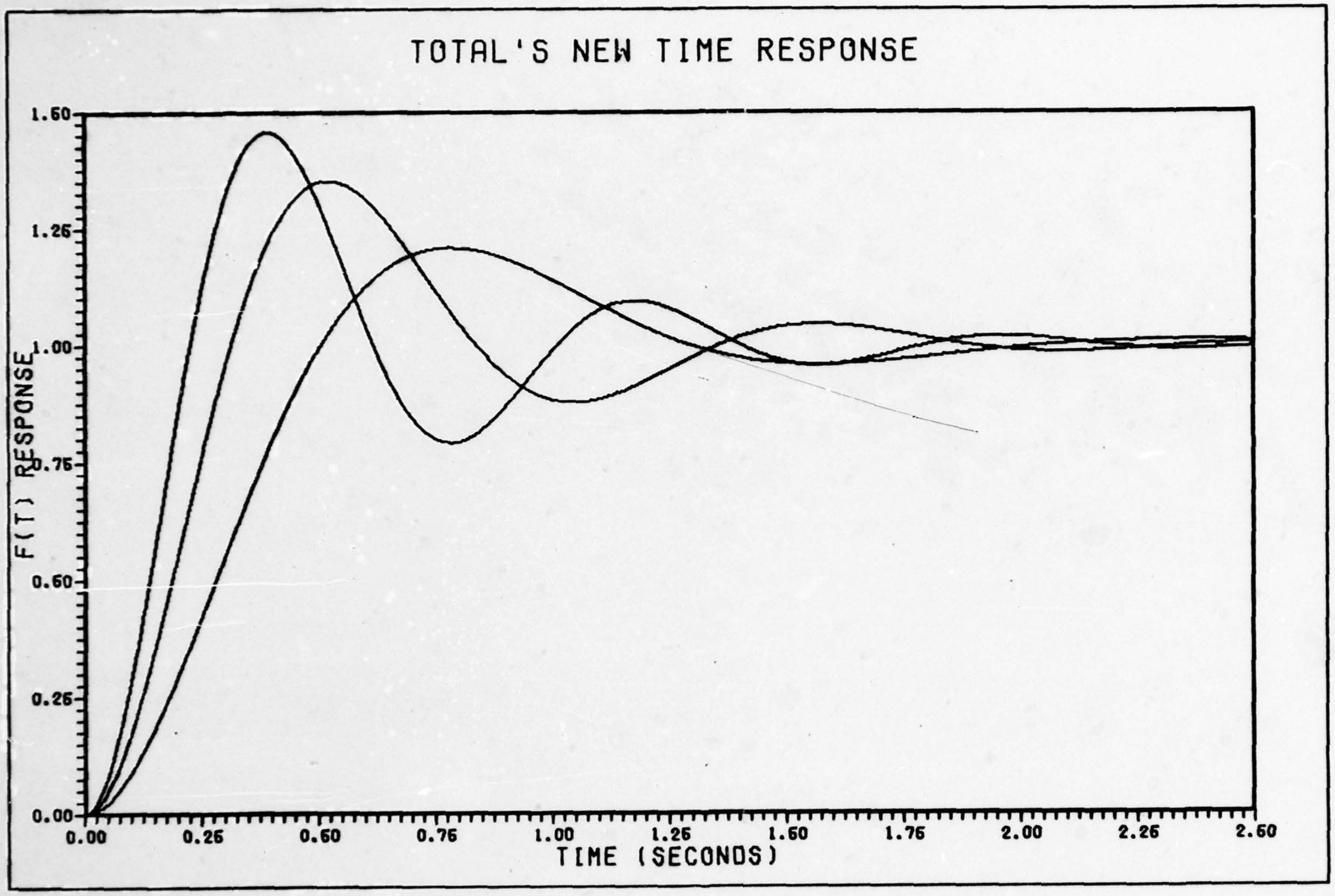
```
> [--------ENTER TITLE  (50 CHARACTERS MAX)--------] <
> SAMPLE TITLE USING ALL 50 CHARACTERS AS AN EXAMPLE
```

If no title is desired the user may simply enter a blank space.

TOTAL also provides the capability for drawing more than one plot on the same set of axes. After MULT, ON has been typed, all subsequent plots are placed on the next set of axes. This will continue until MULT, OFF is typed or until a plot other than a time response (i.e. root locus or frequency response) is generated. When multiple plots are drawn, the F(T) axis is scaled to the maximum and minimum values of the first response plotted. If subsequent plots exceed the axis range clipping will occur, so it is recommended that the largest plot be drawn first (or a large enough axis specified with SCALE, OFF).

The physical size of the plot may be controlled by setting the variable FACTOR equal to the desired scale. Normally, with FACTOR = 1, the plot will be 6 x 9 inches. Typing FACTOR = 2 before selecting the option number will double the size of all future plots. FACTOR = 0.5 will reduce the plot to half size.

It is recommended that the user end TOTAL after every four or five plots and dispose them to the plotter to avoid incurring the wrath of the computer installation operator. (TOTAL can always be restarted without loss of information by


TOTAL'S NEW TIME RESPONSE
F(T) RESPONSE
TIME (SECONDS)
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50

using option 1 to recover all data stored in MEMORY.) Grid lines will be drawn on all plots whenever the switch GRID is ON. (See option 93, Section 2.10)

OPTION 35: Print time or difference equation.

If the system is continuous (TSAMP = 0), option 35 will print the time function as follows:

```
THE TIME FUNCTION IS
F(T)=
            -4.4444      T  EXP(-5.0000      T)
            -1.4815      EXP(-5.0000      T)
             .58561      EXP(-2.0000      T) SIN( 6.0000      *T+ 124.695)
             1.0000      EXP( 0.          T)
```

For discrete systems (TSAMP $\neq$ 0), the difference equation corresponding to CLTF(z) = F(z)/R(z) is printed:

```
F(K)  =  (       0.              )*R(K)
      +  (       0.              )*R(K- 1)   -   (     -1.2000000     )*F(K- 1)
      +  (       6.0000000       )*R(K- 2)   -   (      .25000000     )*F(K- 2)
      +  (       0.              )*R(K- 3)   -   (      .10600000     )*F(K- 3)
```

OPTION 36: Partial fraction expansion of CLTF (or OLTF).

This option performs the partial fraction expansion of a transfer function in s or z providing that the order of the numerator is less than the denominator and there is no more than one repeated pole. For example,

$$\text{CLTF}(s) = \frac{1000(s + 4)}{s(s + 10)^2(s + 2 \pm j6)}$$

expands to

$$\text{CLTF}(s) = \frac{1}{s} + \frac{6}{(s + 10)^2} + \frac{0.56}{(s + 10)} + \frac{-0.78 - 0.29}{s + 2 - j6} + \frac{-0.78 + 0.29}{s + 2 + j6}$$

and is tabulated by option 36 as shown on the next page.

```
                    PARTIAL FRACTION EXPANSION TERMS
      NUMERATOR COEFFICIENTS                    CORRESPONDING POLES          ORDER

(  6.0000      ) + J(   0.          )    ( -10.0     ) + J(   0.       )      2
(  .56000      ) + J(   0.          )    ( -10.0     ) + J(   0.       )      1
( -.78000      ) + J( -.29333       )    ( -2.00     ) + J(  6.00      )      1
( -.78000      ) + J(  .29333       )    ( -2.00     ) + J( -6.00      )      1
(  1.0000      ) + J(   0.          )    (  0.       ) + J(   0.       )      1
```

Note that a pole at the origin due to the presence of a step input was included. Option 39 may be used to select an impulse input if this pole is not desired.

<u>OPTION 37</u>: <u>List time response figures of merit</u>.

Option 37 computes most of the characteristics of a time response that are of interest to the user including rise time (the time to go from 10% to 90% of the final value), duplication time (time from zero to first intersection with the final value), peak time (time to reach <u>highest</u> peak), setting time (last time at which the response was outside a 2% envelope around its final value), peak value (magnitude of highest peak), and the final value of the response as time approaches infinity.

```
OPTION >  37

   CONTINUOUS TIME RESPONSE FOR CLTF(S)
   WITH STEP INPUT OF STRENGTH = 1.

  RISE TIME:           TR=      .180508
  DUPLICATION TIME:    TD=      .300319
  PEAK TIME:           TP=      .522441
  SETTLING TIME:       TS=      2.19580
  PEAK VALUE:          MP=      1.56715
  FINAL VALUE:         FV=      1.00000
```

A sample output for an underdamped system is shown above.

<u>OPTION 38</u>: <u>Quick sketch at user's terminal</u>.

Option 38 is similar to option 32 except that the user can specify initial time ,final time, <u>and</u> the time increment to be plotted (thereby controlling the length and resolution of the plot.) In addition, the plot is automatically scaled to use the entire width of the paper giving maximum resolution, but usually resulting in unusual scale divisions. This option is primarily intended for the user who is only interested in a quick sketch of the general shape of the response curve and not in reading values off the plot.

OPTION 39: Select step, ramp, impulse, pulse, or sine input.

The default input for all time response options is a unit step function, however, the user may select other inputs using option 39. If TOTAL is in continuous mode (TSAMP = 0) when this option is selected the user is asked to input ramp slope, pulse width, etc., in seconds. For discrete mode (TSAMP $\neq$ 0) these variables are input in terms of number of samples.

2.5 ROOT LOCUS OPTIONS

Option 40 through 49 provide the user with a variety of tools for studying the root locus of the open-loop transfer function (OLTF). In order to use this option effectively it is important that the user understand thoroughly the equations and variables involved.

Root Locus Equations

In its simplest sense, the open-loop transfer function is just a ratio of two polynomials in s (or z). For example:

$$\text{OLTF}(s) = \frac{\text{OLNPOLY}(s)}{\text{OLDPOLY}(s)} = \frac{3s^2 + 9s + 6}{2s^3 + 24s^2 + 94s + 120}$$

is a possible open-loop transfer function where OLNPOLY and

OLDPOLY are array names in TOTAL containing the Open-Loop Numerator POLYnomial and Open-Loop Denomiantor POLYnomial coefficients respectively. If these polynomials are factored, the same transfer function can be written as shown below:

$$OLTF(s) = \frac{OLNK \cdot OLZERO}{OLDK \cdot OLPOLE} = \frac{3 \cdot (s + 1)(s + 2)}{2 \cdot (s + 3)(s + 4)(s + 5)}$$

where OLPOLE and OLZERO are complex arrays of pole and zero coordinates in the s-plane. OLNK (the Open-Loop Numerator constant) and OLDK (the Open-Loop Denominator constant) together comprise the total fixed open-loop gain (OLNK/OLDK).

However, the entire concept of root locus centers around varying the overall open-loop gain (static loop sensitivity) and observing how the poles of the closed-loop system move in the s-plane. This static loop sensitivity is defined in TOTAL as

OLK = GAIN * (OLNK/OLDK)

where GAIN is an added gain factor which is varied from zero to infinity along the root locus. Since the system gain (OLNK/OLDK) remains constant, OLK also varies from zero to infinity. In fact, if OLNK/OLDK = 1.0, GAIN = OLK is the static loop sensitivity. The sensitivity at a given point on the root locus is also defined as

$$OLK = \frac{\text{product of distances from locus point to each pole in the system}}{\text{product of distances from locus point to each zero in the system}}$$

Using this definition of OLK, GAIN is just OLK/(OLNK/OLDK). In other words, GAIN is the amount of gain that must be added in the open-loop to the fixed system gain (OLNK/OLDK) to produce the desired closed-loop poles.

General Comments

The root locus is computed one branch at a time within a region of the complex plane which the user can specify. Calculations begin from poles within the region or locus points on the boundaries and proceed until the locus branch leaves the region of calculation or terminates on a zero. Locus points are calcualted DEL units apart and printed every DELPR units. Option 49 describes all of the optional variables which may be used to adjust boundaries, step sizes, and other program functions.

By typing ECHO, OFF (see Section 2.10) the user can suppress the listing of poles and zeros at the beginning of each option. As always, if the switch ANSWER is on, output will be written to the local file ANSWER instead of the user's terminal.

Calcomp Plots

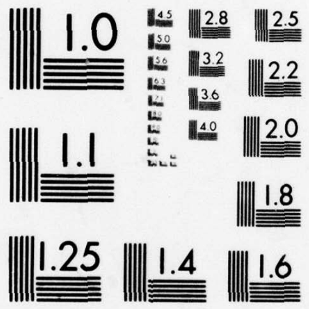
The user can obtain a Calcomp plot of the root locus within the specified region of calculation for options 41, 42, 43, and 48 by typing PLOT, ON prior to selecting the option number. After each plot, the switch PLOT is automatically turned back off to prevent accidental generation of unwanted plots. More than one root locus may be drawn on the same set of axes by typing MULT, ON prior to generation of the first plot. Plots will continue to overlap until MULT, OFF is typed or until a plot other than root locus (i.e., time or frequency response) is generated. Additional switches controlling plot titles, etc. are described in option 93.

Zero-angle Root Locus

The zero-angle root locus is automatically computed instead of the usual 180^o locus whenever OLK is negative.

AD-A055 418 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL..ETC(U)
MAR 78 S J LARIMER
UNCLASSIFIED AFIT/GGC/EE/78-2 N/L

2 OF 3
AD
A055418



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

OLK = GAIN * (OLNK/OLDK) < 0

The user can arrange this by simply setting OLK, GAIN, OLNK, or OLDK equal to a negative number. Similarly, if the system gain (OLNK/OLDK) is negative and the 180^o locus is desired, the user can make OLK positive by making GAIN negative, and so on.

<u>OPTION 40</u>: <u>List options</u>.

This option gives a quick reference list of options 40 through 49.

<u>OPTION 41</u>: <u>General root locus</u>.

This option computes each branch of the open-loop transfer function (OLTF) over a specified bounded region of the complex (s or z) plane. Output consists of a tabular listing of locus points for each branch or sub-branch of the locus and a Calcomp plot (written to file PLOT) if the switch PLOT is on. If z-plane is selected (TSAMP $\neq$ 0) the unit circle will be drawn on the locus plot.

<u>OPTION 42</u>: <u>Root locus with a GAIN of interest</u>.

This option is identical to option 41 except that the user is asked to specify GAIN and GTOL. GAIN is the variable part of the static loop sensitivity OLK:

OLK = GAIN * (OLNK/OLDK)

Option 42 calculates the exact values of the closed-loop roots at the value of GAIN specified. As in 41, a tabular listing of points spaced DELPR units apart is printed, however within a range $\pm$ GTOL of the value of GAIN specified, every point calculated (spaced DEL units apart) is printed.

The user is cautioned that the root calculated at the specified GAIN is only the point at the current calculation step size, DEL, which is closest to the value of GAIN specified. To come closer to the exact value, DEL must be made smaller.

If a Calcomp plot is requested, the roots at the GAIN of interest are marked on the plot.

OPTION 43: Root locus with a damping ratio of interest.

This option is similar to option 42 except that the GAIN of interest is automatically calculated at a value of damping ratio, ZETA, which is of interest. After the value of GAIN is found (by searching along a constant zeta line until an intersection with the locus occurs), option 43 is identical to option 42. The user is asked to input the desired value of ZETA (between 0 and 0.9), RAD, and GTOL where GTOL has the same meaning as in option 42. The ZETA line may interesect the locus more than once. If the further point is desired, RAD (the distance from the origin at which the zeta search starts) should be set large enough to miss the unwanted intersection.

The user is cautioned that if the intersection with the locus occurs outside the specified boundaries, the zeta search (which stops at the boundary) will fail and GAIN will retain its old value. Should this happen, it is only necessary to extend the BB (top) and CC (left) boundaries to include the intersection and try again.

On the Calcomp plot a radial line will be drawn from the origin corresponding to the specified ZETA. The roots at the corresponding GAIN of interest are also marked on the plot.

OPTION 44: <u>List n points on a branch of interest</u>.

This option is intended primarily to investigate parts of the locus which are of special interest. Only one branch of the locus is tabulated, beginning at a specified point given by input variables XSTART and YSTART. The user also specifies the number of points (NPOINTS) to be calculated along the locus at the current step size DEL. No plot is available with this option.

OPTION 45: <u>List all points on a branch of interest</u>.

This is similar to option 44 except that all points on the branch starting at XSTART, YSTART and lying within the specified boundaries are calculated.

OPTION 46: <u>List locus roots at a GAIN of interest</u>.

This option is a truncated version of option 42. The only output is a list of roots at the specified value of GAIN of interest. No plot is generated.

OPTION 47: <u>List locus roots at a ZETA of interest</u>.

This option is a truncated version of option 43. The only output is a list of roots at the specified ZETA of interest. Again, the user is cautioned that if the intersection of the locus and zeta line occurs outside the specified boundaries it will not be found. Extending the BB (top) and CC (left) boundaries will solve this problem.

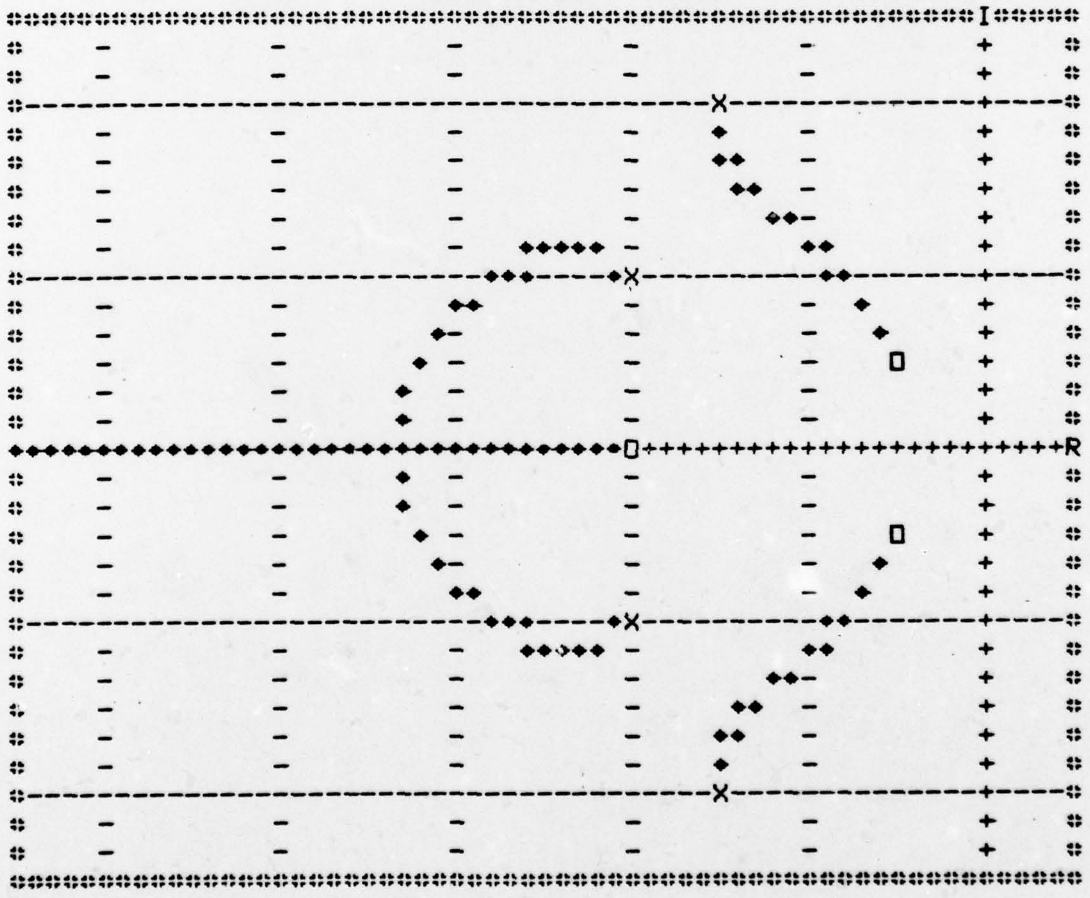
OPTION 48: <u>Plot root locus at user's terminal</u>.

This option produces a printed plot of the root locus within the specified boundaries AA (right, BB (top), CC (left),

```
OPTION >   AA=1 BB=5 CC=-11 DD=-5  48


 OPEN-LOOP (OLTF) ROOT LOCUS USING OPTION 48

   REGION OF CALCULATION-REAL:   CC=   -11.0         TO AA=    1.00
                           IMAJ:   DD=   -5.00         TO BB=    5.00
```

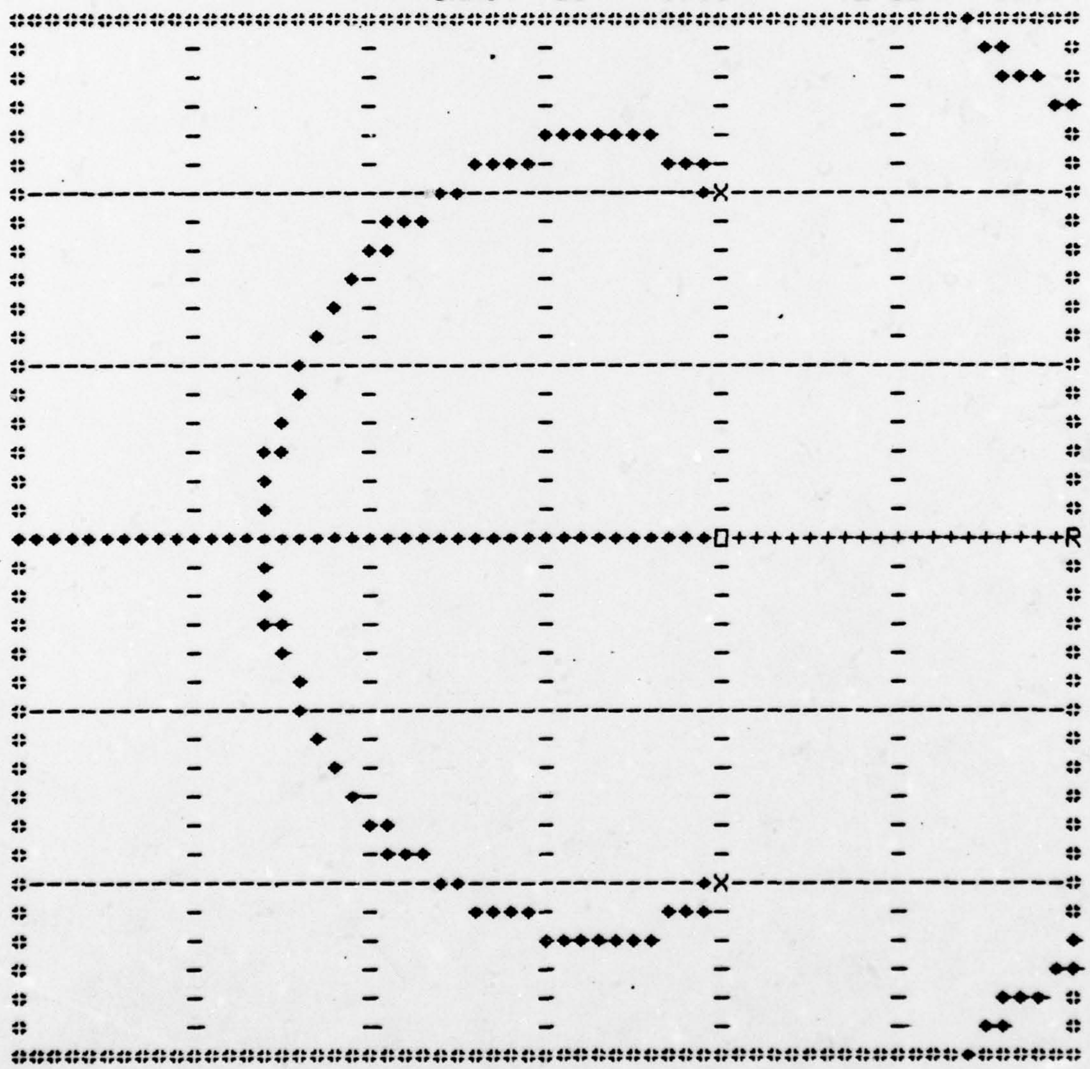


```
   GRID SCALE:   X-AXIS:   1 INCH=        2.0000
                 Y-AXIS:   1 INCH=        2.0000


OPTION >
OPTION >  AA=-2 BB=3 CC=-8 DD=-3  48


OPEN-LOOP (OLTF) ROOT LOCUS USING OPTION 48

  REGION OF CALCULATION-REAL:  CC=  -8.00        TO AA=  -2.00
                        IMAJ:  DD=  -3.00        TO BB=   3.00
```



```
  GRID SCALE:  X-AXIS:  1 INCH=        1.0000
               Y-AXIS:  1 INCH=        1.0000


OPTION >
```

and DD (bottom). Since the plot is always six divisions (six inches) wide, if the user picks the right and left boundaries to be some multiple of six units apart, the plot scale will have nice even divisions. Also, since the locus below the real axis is identical to that above it, the user may often find it advantageous to locate the lower (DD) boundary just below the axis to limit the number of print-out lines produced.

Sometimes when pole and/or zeros are very close together, not all of them will show up on the plot due to limits in resolution. Shrinking the boundaries to include just the area of interest will usually solve this problem. Since poles are always placed on the plot after the zeros, a cancelled pole-zero pair will appear as a pole.

Sample plots produced by option 48 are shown on the preceeding pages. Notice how the boundaries AA, BB, CC, and DD were set prior to typing the option number.

<u>OPTION 49</u>: <u>Print current values of all root locus variables</u>.

Option 49 is designed to help the user remember all of the special-purpose variables and their current values. These variables are optional, but add considerably to the power of the program. They may be specified directly in OPTION mode by typing, for example, "AA = 4.5" or "GAIN = 1000" etc. These variables include:

NAME	DEFINITION	DEFAULT VALUE
AA	Right boundary in the complex plane	+1.0
BB	Top boundary	+3.0
CC	Left boundary	-5.0

NAME	DEFINITION	DEFAULT VALUE
DD	Lower boundary	-3.0
BOUND	A scale factor which multiplies AA, BB, CC, and DD by its value <u>when the next root locus option is executed</u> and then resets itself to BOUND = 1.0.	1.0
ZETA	Damping ratio of interest for which roots are found in options 43 and 47.	0.0
RAD	Distance from the origin at which the zeta intersection search is started.	0.01
GTOL	A region around the gain of interest (GAIN) for which extra locus points are printed. If GTOL = 0, no roots of interest are calculated.	0.0
DEL	Calculation step size between locus points.	0.1
DELPR	Printing step size between locus points.	0.2
FIGURE	If FIGURE ≠ 0, the value of FIGURE will be printed on the Calcomp plot. For example, if FIGURE = 3, the title "FIGURE NUMBER 3" will be drawn. Subsequent plots are numbered 4, 5, etc., until FIGURE is set back to 0.	0.0

2.6 FREQUENCY RESPONSE OPTIONS

The next ten options (50 - 59) perform continuous or discrete frequency response analysis of either open or closed-loop transfer functions. To make fullest use of these options, the user should be aware of several important mode control switches: CLOSED, ANSWER, DECIBEL, DEGREES, HERTZ, and TSAMP. With the exception of TSAMP, these switches are set in OPTION mode by typing "CLOSED, ON" or HERTZ, OFF, etc.

CLOSED selects either the closed-loop transfer function (CLTF) or the open-loop transfer function (OLTF) for analysis. ANSWER controls whether the output comes to the user's terminal or goes to the file ANSWER. When DECIBEL is on, magnitudes are output in decibels, when it is off, linear magnitude is output. Similarly, DEGREES and HERTZ select degrees vs. radians and hertz vs. radians per second respectively. See option 93 for further information on these switches.

TSAMP is not actually a switch in the on-off sense. It is a variable containing the value of the sampling time in seconds. When TSAMP = 0, TOTAL assumes that the system to be analyzed is continuous and expressed in terms of the Laplace operator s. If TSAMP is given some positive value, the system is considered to be discrete in terms of the z-transform operator z with the specified sampling time.

Note that switches must be set while in OPTION mode <u>before</u> typing the desired option number. Switches remain as set until changed by the user and need not be retyped before every option.

As always, prior to selecting any option the user must supply a transfer function using an appropriate input option (options 0 - 9).

<u>OPTION 50</u>: <u>List options</u>.

This option provides a quick reference list of options 50 through 59.

<u>OPTION 51</u>: <u>Tabular listing</u>.

This option tabulates the magnitude and phase angle for a range of frequencies. The user is asked to specify the initial, final, and delta frequencies in hertz or radians per second depending on the switch HERTZ. Units for the tabulated magnitude and phase are dependent upon switches DECIBELS and DEGREES respectively.

OPTION 52: Two-cycle scan of magnitude.

Option 52 tabulates the response magnitude over two cycles (powers of 10) of frequency. Fifteen points per cycle are tabulated at frequency increments designed to space them evenly over the cycle when on a logrithmic scale. The user is asked to specify the power of ten of the starting frequency. For example, if the desired starting frequency was $0.01 = 10^{-2}$, the user would type simply "-2". Option 52 would then tabulate 30 points spaced logrithmically over a range from 0.01 to 1.0. Similarly, typing "1" would specify the range from 10 to 1000.

Again, the units of the variable tabulated depends on switches DECIBELS, DEGREES, and HERTZ. A sample option is shown below:

```
OPTION >  52
  CLOSED-LOOP FREQUENCY RESPONSE USING OPTION 52
ENTER POWER OF STARTING FREQ (-2 FOR .01, ETC) >  0

    W(RAD/SEC)          DECIBELS          W(RAD/SEC)          DECIBELS

    1.0000000          .17446115          10.000000          -5.1188336
    1.2000000          .25167886          12.000000          -9.1381385
    1.4000000          .34327559          14.000000          -12.347703
    1.7000000          .50795432          17.000000          -16.195172
    2.0000000          .70581074          20.000000          -19.294189
    2.5000000          1.1102677          25.000000          -23.427002
    3.0000000          1.6075770          30.000000          -26.732514
    3.5000000          2.1911476          35.000000          -29.493366
    4.0000000          2.8399666          40.000000          -31.866739
    4.5000000          3.5038376          45.000000          -33.949576
    5.0000000          4.0823997          50.000000          -35.806114
    6.0000000          4.3179828          60.000000          -39.007494
    7.0000000          2.6710388          70.000000          -41.705917
    8.0000000       -.61716958E-13        80.000000          -44.038923
    9.0000000         -2.6965885          90.000000          -46.094156

OPTION >
```

<u>OPTION 53</u>: <u>Two-cycle scan of phase angle</u>.

Option 53 is identical to 52 except that phase angle is tabulated instead of magnitude.

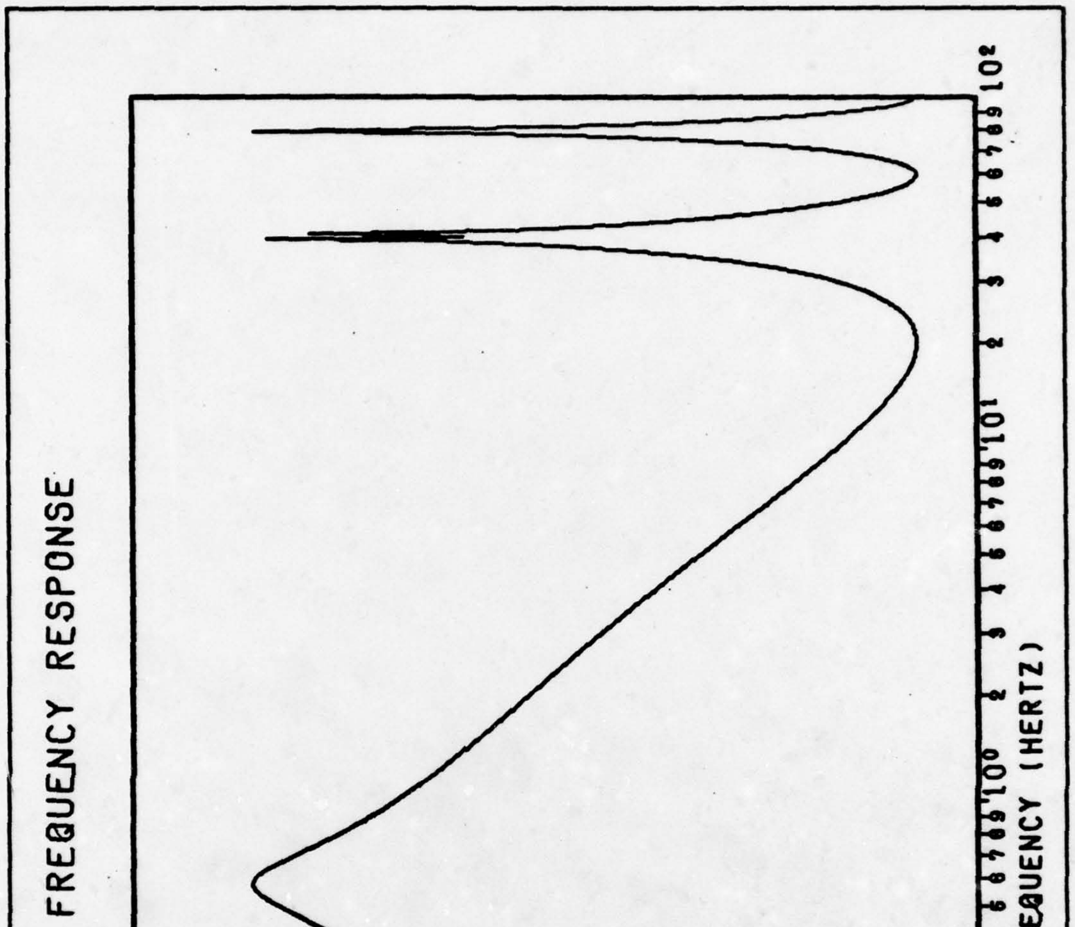
<u>OPTION 54</u>: <u>Plot magnitude or angle at user's terminal</u>.

This option prints out a 6 x 8 inch plot of magnitude or phase angle over a range of frequency specified by the user. The plot axis are scaled identically to those of the Calcomp plots in options 55 and 56. Thus, this option may be used to preview (within limits of printer resolution) exactly how a Calcomp plot would appear. The magnitude axis is normally scaled automatically to easily interpreted dimensions, however, the user may select his own scale when the switch SCALE is off.

<u>OPTION 55</u>: <u>Calcomp plot -- Linear frequency axis</u>.

This option writes a Calcomp plot to a local file PLOT for later disposition to a plotting device. The linear frequency axis feature of this option is intended primarily for discrete frequency response plots where the effects of aliasing are being studied or for plotting small ranges of frequency.

The user is first asked to enter the starting and final frequencies to be plotted. Since, in this option, the frequency axis will have ten divisions on the plot, it is usually wise to select the axis range to be some number evenly divisible by ten. If the switch SCALE is on, the program will automatically scale the magnitude axis. If SCALE is off, the user will be asked to specify the minimum and maximum axis values. Again, since the magnitude axis is always six divisions long, the axis


DISCRETE FREQUENCY RESPONSE
MAGNITUDE (DECIBELS)
30
20
10
0
-10
-20
-30
FREQUENCY (HERTZ)
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}

length should be chosen to be evenly divisible by six for best results.

The user is also given a choice for a plot of phase, magnitude, or both on the same plot.

A plot title of maximum length 50 characters is also requested.

OPTION 56: Calcomp plot -- Log frequency axis.

This option is similar to option 55 except the frequency axis is logrithmic with some integer number of logrithmic cycles. The user is asked to input the power of the starting frequency (-2 for 0.01, 3 for 1000, etc.) and the number of cycles to be plotted (maximum of ten). As in option 55, the plot may be in magnitude or decibels, degrees or radians, and open or closed-loop depending on switch settings. Typing GRID, ON will place grid lines on the Calcomp plot as with the time response plots.

OPTION 57: Tabulate points of interest.

This option is similar to option 37. It finds and tabulates all response peaks, zero-db crossings, 180 degree crossings, break frequencies, and asymtotes for the given transfer function.

OPTION 58: Calcomp plot -- Nyquest polar plot.

This option produces a polar plot of magnitude vs. angle with frequency as a parameter.

OPTION 59: Calcomp plot -- Nichol's log magnitude/angle plot.

Option 59 produces a semi-log plot of magnitude vs. phase angle with frequency as a parameter.

2.7 POLYNOMIAL OPERATIONS

There is a large class of control problems which requires extensive manipulation of polynomials and their associated first-order factors. Options 60 through 69 have been designed to relieve the engineer of the extensive hand calculations which would otherwise be necessary to solve these problems. For complete understanding of these options, a few words of explanation are necessary.

TOTAL has twelve general purpose polynomial arrays in which the user can store polynomials of maximum degree 50. The names by which the user can refer to these arrays are POLYA, POLYB, POLYC, POLYD, GNPOLY, GDPOLY, HNPOLY, HDPOLY, OLNPOLY, OLDPOLY, CLNPOLY, and CLDPOLY. POLYA through POLYD are scratch registers used by options 61 through 69 for polynomial arithmetic. The remaining eight polynomials are paired to form four transfer functions GTF, HTF, OLTF, and CLTF and are accessed by options throughout TOTAL.(See Section 4.5)

Corresponding to twelve polynomials are twelve n x 2 arrays of polynomial roots where the first dimension is the root number and the second designates either the real or imaginary part of the root. The names by which the user can refer to these corresponding root arrays are: ROOTA, ROOTB, ROOTC, ROOTD, GZERO, GPOLE, HZERO, HPOLE, OLZERO, OLPOLE, CLZERO, and CLPOLE, respectively.

The important thing to remember is that the polynomial and root arrays form inseparable pairs -- changing one will change the other. For example, changing a coefficient in POLYB will automatically change the roots in ROOTB. Similarly, changing a root in ROOTD will result in new coefficients in POLYD, and so on. Because of this direct relationship, the polynomials and their roots are always printed out together as shown below:

```
                              CLTF(S) DENOMINATOR
I          CLDPOLY(I)                                   CLPOLE(I)
1      (   .6498       )S** 3        (  -.1261     ) + J(    .6445     )
2      (   .4086       )S** 2        (  -.1261     ) + J(   -.6445     )
3      (   .3420       )S** 1        (  -.3767     ) + J(    0.        )
4      (   .1056       )                          CLDK=     .6498
```

The polynomial shown in this example would read

$$CLDPOLY(s) = 0.6498s^3 + 0.4086s^2 + 0.3420s + 0.1056$$

and its corresponding factors as

$$= 0.6498(s + 0.1261 - j0.6445)(s + 0.1261 + j0.6445)(s + 0.3767)$$

Note that the polynomial constant CLDK is equal to the coefficient of the highest power of s and is simply the constant that must be multiplied with all the factors to yield the original coefficients of the polynomial.

Input of Polynomials

When the user selects options 61 through 65 he will receive the prompt:

ENTER POLYA & POLYB DEGREES (OR SOURCE) >

If he wants to type in the coefficients, he simply enters the degree of each polynomial and TOTAL will ask him for the required coefficients. On the other hand, if one or both of the polynomials he wishes to enter are already stored in one of the twelve polynomial arrays, he simply types the name of that array in place of the polynomial degree in which case TOTAL will get the needed information from that "source". The user may mix names and numbers as needed. In fact, since the names POLYA and ROOTA refer to simply two forms of the same polynomial, typing either name will transfer the same information. The following examples show possible responses to the same prompt:

```
ENTER POLYA & POLYB DEGREES (OR SOURCE) >   3,4
ENTER POLYA & POLYB DEGREES (OR SOURCE) >   POLYB, GNPOLY
ENTER POLYA & POLYB DEGREES (OR SOURCE) >   POLYC, 4
ENTER POLYA & POLYB DEGREES (OR SOURCE) >   CLDPOLY, ROOTB
ENTER POLYA & POLYB DEGREES (OR SOURCE) >   POLYA, GZERO
```

Notice that if, as in the last example, the user wants to leave the contents of POLYA unchanged, he simply types "POLYA" in the appropriate spot.

After each polynomial is entered, it is echoed back to the user's terminal. To suppress this listing, the switch ECHO should be turned off (see option 93, Section 2.10)

OPTION 60: List options.

Option 60 provides a quick reference list of options 60 through 69.

OPTION 61: Factor a polynomial.

If the user is only interested in finding the roots of a polynomial, option 61 is the fastest way. The user simply enters the polynomial degree and its coefficients as requested. Afterwards, the polynomial and its roots remain stored in POLYA and ROOTA, respectively for future use.

OPTION 62: Add polynomials.

This option adds POLYA to POLYB, factors the result, and stores the coefficients and roots in POLYC and ROOTC respectively.

OPTION 63: Subtract polynomials.

Option 63 is identical to 62 except that POLYB is subtracted from POLYA. The results are stored in POLYC and ROOTC for future use.

OPTION 64: Multiply polynomials.

This option stores the product of POLYA and POLYB to POLYC. POLYC is then factored and its roots are stored in ROOTC. This option is automatically aborted if the degree of the resulting polynomial exceeds 50.

OPTION 65: Divide polynomials.

Option 65 divides POLYA by POLYB and stores the quotient polynomial in POLYC. If the division has a remainder, the user is asked to enter a limit on the number of remainder terms to be listed and the division continues into negative powers of s until the division is complete or the limit is reached. As always, POLYC is automatically factored into ROOTC.

OPTION 66: Store any polynomial to POLYD.

POLYD serves as a scratch polynomial for storing intermediate results. For example, if two pairs of polynomials are to be multiplied and the results summed, POLYD could be used to hold the first product while the second product was being formed. Option 66 simply asks the user for the name of the polynomial to be stored and then copies it into POLYD. The contents of POLYD may be recovered at any time by typing POLYD instead of the degree when it is requested. (See "Input of polynomials" in the introduction to Section 2.7)

The COPY command may be used in place of this option for more versitile manipulation of information. For example, typing COPY, POLYA, POLYC when in OPTION mode will copy the contents of POLYA into POLYC. See Section 3 for more information on this command.

<u>OPTION 67</u>: <u>Expand roots into POLYA</u>.

If the user has a collection of first order factors which he wishes to expand into an n^{th} order polynomial, he should use this option. After entering the polynomial degree (number of factors) he will be asked to enter real and imaginary parts of each root. Only one root may be typed per line and both real and imaginary parts must be specified as x-y coordinates in the complex plane. Conjugates, if any, are automatically assumed, and each root is assigned an index number for future reference. The resulting n^{th} order polynomial and its roots are stored in POLYA and ROOTA respectively.

<u>OPTION 68</u>: <u>Expand $(s + a)^n$ into POLYA</u>.

Option 68 provides a quick method of expanding a multiple order real root of the form $(s + a)^n$ into an n^{th} order polynomial. The user is asked to input a and n and the result is stored in POLYA.

OPTION 69: Activate polynomial calculator.

For the individual who has more than one polynomial operation to perform, option 69 is by far the recommended procedure. This option is simply a three function calculator which adds, subtracts, and multiplies polynomials. POLYA, POLYB, POLYC, and POLYD are used as a four-register stack of polynomials which can be "rolled", "exchanged", and manipulated as single numbers are handled in any reverse polish notation calculator.

Operations always occur between POLYA and POLYB where POLYA is the bottom or "X" register of the stack. Typing a "+", for example, adds POLYA to POLYB and leaves the results in POLYA. The contents of POLYC then drop into POLYB and POLYD is copied into POLYC. The contents of POLYD remain unchanged. Similarly, typing a "-" subtracts POLYA from POLYB, stores the results in POLYA, and drops the stack as described above. To multiply POLYA and POLYB a "*" is typed.

The user enters polynomial calculator mode by selecting option 69 and receives the prompt

COMMAND OR DEGREE:

whereupon he may type EX, ROLL, LIST, STACK, +, -, *, $, the name of one of TOTAL's twelve polynomials, or any number from 0 to 50. The meaning of these commands is described below:

The numbers 0 to 50 refer to the degree of a polynomial the user wishes to enter. After such a command, the user is asked to enter the appropriate coefficients and the new polynomial is placed in POLYA. What had been in POLYA gets moved into POLYB, shoving POLYB into POLYC and POLYC into POLYD. The former contents of POLYD are lost.

The user may also enter the current contents of any of <u>TOTAL's twelve polynomials</u> by simply typing the correct name (as listed in the introduction to section 2-7). The specified polynomial is automatically shoved into POLYA at the bottom of the stack, and the contents of POLYD shoved out the top of the stack and lost as described above.

<u>EX</u> is the command to exchange the contents of POLYA and POLYB. As with all operations in option 69, the corresponding root arrays ROOTA and ROOTB are also exchanged.

<u>ROLL</u> is the command to rotate the contents of the stack down and around. The following transfers occur: POLYB to POLYA, POLYC to POLYB, POLYD to POLYC, and the original contents of POLYA to POLYD. Note that no information is lost.

<u>LIST</u> lists the current contents of POLYA and the corresponding array of roots ROOTA.

<u>STACK</u> lists all four polynomials in the stack in four columns with the highest power coefficients on top.

To end polynomial calculator mode, the user types a $.

An example of various calculator operations is given on the following page. Items typed by the user are underlined. The user may suppress the listing of commands at the beginning by typing 69, S when selecting the option.

```
OPTION > 69

STACK LEVELS:     COMMANDS--MEANINGS:
                     EX      --EXCHANGE POLYA & POLYB
   POLYD             ROLL    --ROLL STACK DOWN
   POLYC             + - *   --OPERATIONS
   POLYB             LIST    --LIST POLYA CONTENTS
   POLYA             STACK   --DISPLAY STACK CONTENTS
                     $       --END OPTION 69

COMMAND OR DEGREE: 4

ENTER 5 POLYA COEFF--HI TO LO:
> 12 23 34 45 56


COMMAND OR DEGREE: 3

ENTER 4 POLYA COEFF--HI TO LO:
> 1 3 5 7


COMMAND OR DEGREE: 5

ENTER 6 POLYA COEFF--HI TO LO:
> 100 200 300 400 500 600


COMMAND OR DEGREE: STACK

  POWER     POLYA          POLYB          POLYC          POLYD

    5       100.0
    4       200.0                         12.00
    3       300.0          1.000          23.00           0.00
    2       400.0          3.000          34.00           0.00
    1       500.0          5.000          45.00           0.00
    0       600.0          7.000          56.00           0.00

COMMAND OR DEGREE: +

COMMAND OR DEGREE: STACK

  POWER     POLYA          POLYB          POLYC          POLYD

    5       100.0
    4       200.0          12.00
    3       301.0          23.00           0.00           0.00
    2       403.0          34.00           0.00           0.00
    1       505.0          45.00           0.00           0.00
    0       607.0          56.00           0.00           0.00

COMMAND OR DEGREE: $
```

2.8 MATRIX OPERATIONS

Another large class of control problems require the individual to add, subtract, multiply, invert, transpose, and obtain the determinants of constant coefficients matrices. Options 70 through 79 are designed to facilitate this process by removing some of the more laborous hand computations involved.

To use these options, the user must first supply TOTAL with the necessary matrices using options 10 through 19. Once these matrices (of maximum size 10 x 10) have been stored in some of seven arrays (AMAT, BMAT, CMAT, DMAT, KMAT, FMAT, or GMAT), the user is free to manipulate them using the options about to be described.

OPTION 70: List options.

This option provides a quick reference list of options 70 through 79.

OPTION 71: Compute eigenvalues of AMAT.

The eigenvalues of AMAT are simply the roots of the characteristic polynomial which is defined as the determinant

det(sI - AMAT)

Option 71 simply evaluates the above determinant as a polynomial in s and stores the result in POLYA. POLYA is then factored and its roots, which are the eigenvalues of AMAT, are stored in ROOTA.

OPTION 72: Add matrices.

This option adds AMAT to BMAT and stores the result in CMAT. Naturally, AMAT and BMAT must have the same row and column dimensions for this operation to be defined.

<u>OPTION 73</u>: <u>Subtract matrices</u>.

Option 73 is identical to 72 except that BMAT is subtracted from AMAT.

<u>OPTION 74</u>: <u>Multiply matrices</u>.

Option 74 performs the matrix multiplication

(CMAT) = (AMAT) · (BMAT)

For this operation to be defined, AMAT and BMAT must conform. That is, the number of columns in AMAT must equal the number of rows in BMAT. If AMAT is an NA x MA matrix, and BMAT is NB x MB, where MA = NB, then CMAT will be an NA x MB matrix.

<u>OPTION 75</u>: <u>Inverse of AMAT</u>.

This option calculates the inverse of AMAT and stores the result in CMAT. If AMAT is singular, an error message is printed.

<u>OPTION 76</u>: <u>Transpose of AMAT</u>.

The transpose of a matrix is formed by interchanging its rows and columns so that, for example, a 3 x 5 matrix becomes a 5 x 3 matrix. This option transposes AMAT and stores the result in CMAT.

<u>OPTION 77</u>: <u>Identity matrix</u>.

The identity matrix is a square matrix with ones on the diagonal and zeros elswhere. This option sets CMAT equal to a n x n identity matrix where n is specified by the user. This identity matrix may then be moved to any other matrix where it is needed using the COPY command described in option 79.

<u>OPTION 78</u>: <u>Zero matrix</u>.

This option may be used to set all the elements of DMAT equal to zero. The user can then zero any other matrix by copying the zeroed DMAT into it using the COPY command described in option 79.

OPTION 79: Copy one matrix to another.

Option 79 prints a brief message telling how to use the COPY command.

The COPY command has the following form

COPY, from, to

where "from" is the name of the matrix to be copied and "to" is the name of the matrix to recieve the data.

For example,

OPTION > COPY, AMAT, FMAT

will dimension FMAT to the same size as AMAT and store each element of AMAT to the corresponding element of FMAT. AMAT is unaffected by this operation. Notice that the COPY command is typed when in OPTION mode.

In addition to the seven matrix names already mentioned there are 19 additional locations in mass storage on the local file MEMAUX. This auxiliary memory file brings the total number of addressable matrices to 26 -- one for each letter in the alphabet.

*AMAT	*FMAT	*KMAT	PMAT	UMAT	ZMAT
*BMAT	*GMAT	LMAT	QMAT	VMAT	
*CMAT	HMAT	MMAT	RMAT	WMAT	
*DMAT	IMAT	NMAT	SMAT	XMAT	
EMAT	JMAT	OMAT	TMAT	YMAT	

The names marked * are working registers which may be directly modified and manipulated. They are stored in local file MEMORY (see option 1). The remaining names in the table reside in MEMAUX and are only accessable with the COPY command (although they may be listed at the user's terminal as usual by typing their names). These matrices may be used as auxiliary storage locations whenever needed. For example, the following commands are legal:

```
COPY, AMAT, XMAT
COPY, PMAT, QMAT
COPY, ZMAT, CMAT
```

See Section 3 for further information on COPY and other commands.

2.9 DIGITIZATION OPTIONS

The next ten options (80-89) provide the user with several means of transforming back and forth between digital and continuous domains. Both transfer function and state-space techniques are provided.

All transfer function transformations are performed in-place on the transfer function CLTF. TOTAL does not keep track of whether the current CLTF is a function of s or z, it is simply treated as an array of numbers. If an s to z transformation is performed, TOTAL assumes that CLTF is a function of s. For z to s transformations, CLTF is treated as a function of z. In all cases, for simplicity, when the contents of CLTF are listed, the dummy variable s will be used to represent either the Laplace s or the z-transform z. The user is assumed to be capable of the mental gymnastics necessary to substitute z for s when necessary.

The state-space transformations provided are between AMAT and BMAT (the continuous system and input matrices) and FMAT and GMAT (the discrete system and input matrices). All four of these matrices can exist in TOTAL at the same time in contrast to the in-place transformations between CLTF(s) and CLTF(z). In the latter case, the user may want to copy the original contents of CLTF into some other location using the COPY command, before they are modified with a transformation option.

<u>OPTION 80</u>: <u>List options</u>.

This option provides a quick reference list of options 80 through 89.

<u>OPTION 81</u>: <u>CLTF(s) to CLTF(z) by impulse invariance</u>.

The concept of impulse invariance is based on the fact that if CLTF(s) is expanded into partial fractions

$$\text{CLTF}(s) = \sum_{k=1}^{N} \frac{A_k}{s - s_k}$$

where, in general, s_k is a complex number

$s = a + jb$

then the corresponding impulse invariant z transfer function is just

$$\text{CLTF}(z) = \sum_{k=1}^{N} \frac{A_k}{1 - e^{s_k T} z^{-1}} = \sum_{k=1}^{N} \frac{A_k z}{z - e^{s_k T}}$$

where T is the sampling time.

In other words, a pole at $s = s_k$ in the s-plane transforms to a pole at $e^{s_k T}$ in the z-plane and the coefficients A_k of the partial fraction expansion are equal.

Option 81 simply performs a partial fraction expansion of CLTF(s), moves the poles from s_k to $e^{s_k T}$, and multiplies the resulting terms back together to form CLTF(z).

The algorithm used does not allow repeated poles. An approximate transform is still possible, however, if the repeated poles are separated slightly before transformation.

OPTION 82: CLTF(s) to CLTF(z) by first-difference transform.

The first difference transformation involves the simple substitution

$$s = \frac{1}{T} \cdot \frac{(z - 1)}{z}$$

where T is a given sampling rate. Option 82 makes this substitution into CLNPOLY(s) and CLDPOLY(s) and then reduces the result to a simple ratio of two polynomials in z (CLNPOLY(z) and CLDPOLY(z)).

OPTION 83: CLTF(s) to CLTF(z) by Tustin transform.

This option is similar to 82 except that the substitution made is

$$s = \frac{2}{T} \cdot \frac{z - 1}{z + 1}$$

OPTION 84: CLTF(z) to CLTF(s) by impulse invariance.

This option is just the inverse of option 81. CLTF(z) is expanded in partial fractions, the poles moved from z_k to $(1/T)\ln(z_k)$, and the resulting fractions multiplied together to form CLTF(s). Because of the algorithm used this option will only work for z transfer functions which have at least one zero at the origin of the z-plane.

OPTION 85: CLTF(z) to CLTF(s) by inverse first difference.

This option is just the inverse of option 82 and involves the substitution

$$z = \frac{s}{s - T}$$

where T was the sampling rate used to form the z-transfer function.

OPTION 86: CLTF(z) to CLTF(s) by inverse Tustin transform.

This option is just the inverse of option 83 and involves the substitution

$$z = \frac{s + T/2}{s - T/2}$$

where T was the sampling rate used to form the z-transfer function.

OPTION 87: Find FMAT and GMAT from AMAT and BMAT.

Option 87 uses a truncated power series method to approximate FMAT (the discrete system matrix) and GMAT (the discrete input matrix) from the continuous counterparts AMAT and BMAT. The iterations are continued until an accuracy of ten significant figures is obtained for all elements of each matrix. This method may therefore be considered, for practical purposes, to be exact.

OPTION 88: Compute FMAT = exp(AMAT*T).

This option computes the component matrices of the state transition matrix e^{AT} using the Cayley-Hamilton Theorem (See Shaum's Outline State Space and Linear Systems, pp 101-102) and sets FMAT = exp(AMAT*T) for a value of T specified by the user.

OPTION 89: General transformation.

This option is a generalized form of the class of transformation performed in options 82, 83, 85, and 86. It involves the generalized substitution

$$s = \text{ALPHA} \cdot \frac{z + A}{z + B} \quad \text{or} \quad z = \text{ALPHA} \cdot \frac{s + A}{s + B}$$

where ALPHA, A, and B are constants which the user can define to perform any bilinear transformation which may be of interest.

2.10 MISCELLANEOUS OPTIONS

The remaining options perform a variety of functions, some of which are of particular interest.

OPTION 90: List options.

This option gives a quick reference list of options 90 through 99.

OPTION 91: Store all data in TOTAL to local file MEMORY.

This option is the direct counterpart to option 1. Normally, all data is automatically stored in MEMORY when the user ends TOTAL by typing STOP. Option 91 allows the user to store this data without terminating the program. It is recommended that the user use this option from time to time to prevent loss of all information in TOTAL in the event of an abnormal termination. (Such an occurance is supposed to be impossible with TOTAL, but nothing is ever 100% fool proof.) See option 1 for further information on the use of MEMORY.

OPTION 92:

This is a spare option which was not defined at the time this edition of the user's manual was prepared.

OPTION 93: List switch settings.

This option prints out the current values of all switch settings in TOTAL. Typing HELP, 93 gives a complete description

```
OPTION >  93

TOTAL'S MODE CONTROL SWITCHES ARE SET AT:

ECHO:      OFF           CLOSED:    ON
ANSWER:    OFF           HERTZ:     OFF
PLOT:      OFF           DEGREES:   ON
TITLE:     OFF           DECIBELS:  ON
CAPTION:   OFF           MULT:      OFF
GRID:      OFF           SCALE:     ON

FOR AN EXPLANATION OF THESE SWITCHES AND HOW TO SET THEM
TYPE:  HELP,93

OPTION >  HELP,93

TOTAL HAS 10 SWITCHES WHICH CONTROL ITS PERFORMANCE.
THE USER MAY SET THESE SWITCHES WHEN IN OPTION MODE BY
TYPING:    SWITCHNAME,ON   OR   SWITCHNAME,OFF

THE SWITCHES AND THE FUNCTIONS THEY CONTROL ARE:

ANSWER    ,ON  CAUSES ALL OUTPUT TO GO TO FILE ANSWER
          ,OFF DISPLAYS OUTPUT AT USER'S TERMINAL
ECHO      ,ON  INPUT IS ECHOED BACK TO THE TERMINAL
          ,OFF ECHO OF INPUT IS SUPPRESSED
CLOSED    ,ON  CLOSED-LOOP (CLTF) USED IF THERE'S A CHOICE
          ,OFF OPEN-LOOP TRANSFER FUNCTION (OLTF) IS USED
HERTZ     ,ON  FREQUENCY IS INPUT AND OUTPUT IN HERTZ
          ,OFF FREQUENCY IS IN RADIANS/SEC
DECIBELS  ,ON  MAGNITUDES ARE IN DECIBELS=20*ALOG10(MAG)
          ,OFF ACTUAL MAGNITUDE IS OUTPUT
TITLE     ,ON  AN ADDITIONAL TITLE TO BE DRAWN OUTSIDE THE
               BORDER WILL BE REQUESTED WHEN NEXT ROOT LOC
               PLOT IS DONE.  TITLE WILL BE DRAWN ON ALL
               FUTURE PLOTS UNTIL  TITLE,OFF  IS TYPED.
CAPTION   ,ON  A 3-LINE CAPTION WILL BE REQUESTED AND
               DRAWN WHEN NEXT ROOT LOCUS IS GENERATED.
          ,OFF THE OPEN-LOOP TRANSFER FN. IS DRAWN ON
               ALL ROOT LOCUS PLOTS.
PLOT      ,ON  GENERATES CALCOMP PLOT FOR OPTIONS 41,42,43,& 48
          ,OFF NO PLOT (PLOT IS AUTOMATICALLY TURNED BACK
               OFF AFTER EACH PLOT)
MULT      ,ON  FUTURE PLOTS ALL DRAWN ON NEXT AXIS SET
          ,OFF EACH PLOT IS DRAWN SEPARATELY
GRID      ,ON  DRAW GRID LINES ON PLOTS
          ,OFF OMIT GRID LINES ON PLOTS
SCALE     ,ON  AUTO SCALE CALCOMP PLOTS
          ,OFF LET USER SPECIFY SCALE

NOTE:  IF A SWITCHNAME IS TYPED WITHOUT ,ON  OR ,OFF,
       THE SWITCH IS TURNED ON.

OPTION >
```

of each switch and how to use them. An example of both is given on the following page.

<u>OPTION 94</u>:

This is a spare option as yet undefined.

<u>OPTION 95</u>:

This option at the time of this writing is as yet undefined.

<u>OPTION 96</u>: <u>List special commands</u>.

This option gives a brief list of the special commands that are allowed in OPTION mode. See Section 3 for a complete description of these commands. This same list can be generated by typing <u>COMMANDS</u>.

<u>OPTION 97</u>: <u>List variable name directory</u>.

This option is designed to serve as a memory aid for the user who is trying to remember a particular variable name. These variables, and how to list and modify them are discussed in detail in Section 4. This same list of variables may be obtained at any time by typing <u>VARIABLES</u>.

<u>OPTION 98</u>: <u>List main options</u>.

This option lists the ten main option groups in TOTAL. The same list may be obtained by typing the command: <u>OPTIONS</u>.

<u>OPTION 99</u>: <u>Give the introduction to TOTAL</u>.

This option gives a brief introduction to TOTAL. It is the same introduction obtained when the user types <u>HELP</u>.

SECTION 3. SPECIAL COMMANDS

TOTAL has a number of special commands which are designed to enhance the user's control of the program. In the list below, items in capital letters are actual commands, items in parenthesis are optional parameters, and those in small letters refer to general command types. Underlined letters indicate the minimum abbreviation which is allowed for each item. Commands may by typed only in OPTION mode. (See Section 1.1)

The following are the special commands:

STOP, (SUP)	End program (suppress messages)
HELP, (option number)	Get help on (specified option number)
Swtichname, (ON or OFF)	Turn specified switch (on or off)
COPY, from, to	Copy one variable to another
PAGE	Skip to top of new output page
DELETE, root(I), (VETO)	Remove a specified pole or zero (display its value first)
CALCULATOR	Delayed entry to calculator mode
CREATE, keyname, string	Define a macro with the specified keyname to execute the specified string of commands.

STOP, (SUP).

This command stores all data into local file MEMORY, writes messages to notify the user of any files that have been created during execution, and terminates TOTAL. These messages are suppressed if the user types "STOP, S".

HELP, (option number).

The user may obtain help on any option by simply typing HELP followed by the option number of interest. For example "HELP, 38" will give a brief description of option 38.

Switchname, (ON or OFF).

This command allows the user to set TOTAL's twelve mode control switches which include ECHO, ANSWER, PLOT, TITLE, CAPTION, GRID, CLOSED, HERTZ, DEGREES, DECIBELS, MULT, and SCALE. These switches allow the user to custom tailor TOTAL's performance to his individual preference. See option 93 in Section 2.10 for further details.

COPY, from, to.

The COPY command is a special utility which allows the user to transfer the contents of entire arrays from one location to another. COPY may be used to copy transfer functions, polynomials, or matrices to other transfer functions, polynomials, or matrices respectively. The program will ignore attempts to copy a transfer function into a matrix, etc.

Transfer Functions. TOTAL has 28 transfer function storage locations of which GTF, HTF, OLTF, and CLTF are working registers stored on local file MEMORY and the remainder are auxiliary registers stored on local file MEMAUX and are accessable only with the COPY command. These transfer functions include:

ATF	ETF	ITF	MTF	QTF	UTF	YTF
BTF	FTF	JTF	NTF	RTF	VTF	ZTF
CTF	*GTF	KTF	OTF	STF	WTF	*OLTF
DTF	*HTH	LTF	PTF	TTF	XTF	*CLTF

Transfers are legal between any two of the above locations. For example, COPY, GTF, XTF will transfer GTF into XTF. GTF is unaffected.

Polynomials. Transfer is also permitted between any two of TOTAL's twelve polynomials including POLYA, POLYB, POLYC, POLYD, GNPOLY, GDPOLY, HNPOLY, HDPOLY, OLNPOLY, OLDPOLY, CLNPOLY, and CLDPOLY. For example, COPY, POLYA, CLNPOLY will transfer the contents of POLYA into CLNPOLY.

Matrices. Finally, TOTAL has 26 matrix locations which may be addressed by the copy command. These matrices include:

*AMAT	EMAT	IMAT	MMAT	QMAT	UMAT	YMAT
*BMAT	*FMAT	JMAT	NMAT	RMAT	VMAT	ZMAT
*CMAT	*GMAT	*KMAT	OMAT	SMAT	WMAT	
*DMAT	HMAT	LMAT	PMAT	TMAT	XMAT	

Transfer is allowed between any two locations. For example COPY, AMAT, QMAT copies AMAT to QMAT.

PAGE.

This command is primarily for use when the switch ANSWER is on. It places a "1" in column 1 on the file ANSWER as a carriage control character for a line printer. This causes the printer to skip to the top of a new computer sheet before continuing the printout. The PAGE command is particularly useful for separating output between different problems and any time a new sheet of paper is desired. To execute this command the user simply types "PAGE".

DELETE, root(I), (VETO).

The user may modify the location of a pole or zero at any time by simply typing its name, an equal sign, and its new value. A pole or zero can be added in the same manner by simply naming the root with the next highest index number. (See Section 4.3) However, to remove a root and thereby reduce the order of the system, the DELETE command is required. For example typing "DELETE, CLPOLE(3)" will remove pole number 3 from the closed-loop transfer function (CLTF), reduce the order of the system by one, resequence the index numbers on the remaining poles, and recalculate the coefficients in CLDPOLY. If CLPOLE(3) has a complex conjugate, both roots will be deleted and the system order reduced by two. If the user does not know the index number of the root he wants to delete, he can type the array name for a list. For example, typing "CLPOLE" would list all pole locations and their corresponding index numbers.

The VETO command is an optional suffix. If "DELETE, CLPOLE(3), VETO" were typed, the value of CLPOLE(3) would be printed and the user asked to type "<u>Y</u>ES" or "<u>N</u>O" to delete or not. This feature allows the user to confirm that he has selected the right root before completing the deletion. The user may abbreviate the command as "D, CLPOLE(3), V" if desired.

<u>CALCULATOR</u>.

This command allows delayed entry into CALCULATOR mode. Calculator is normally entered by typing a simple "C". This causes CALCULATOR mode to begin immediately when the carriage return button is pressed, regardless of how many commands preceed it in the string. For example,

```
    OPTION >   ECHO, ON  38  AA=4.5  C DD=19
```

will be executed in this order:

```
    C   ECHO,  ON   38  AA=4.5    DD=19
```

This is to allow the user to jump directly to calculator mode at any time. If, however, the user wants to enter calculator in the sequence he specified, he must use an abbreviation longer than "C", such as "CALC". For example,

OPTION > <u>ECHO</u>, <u>ON</u> <u>38</u> <u>AA=4.5</u> <u>CALC</u> <u>DD=19</u>

<u>CREATE, keyname, commandstring</u>.

Using CREATE, the user can define his own macro command made up of any combination of option numbers, commands, variable names and other macros which he uses frequently. In other words, this command makes TOTAL programable.

Three macro command names (key names) are allowed: AKEY, BKEY, and CKEY. Each key can be made of up to 50 instructions to do anything allowed in OPTION mode. These instructions are then executed like a subprogram any time the key name is typed.

For example, if the user wants to write a key which will turn ANSWER on, change the value of a pole to the current value in the X register of the calculator, and plot a root locus using option 48, he can type:

<u>CREATE</u>, <u>AKEY</u>, <u>ANSWER</u>, <u>ON</u> <u>OLPOLE(2)=X</u> <u>48</u>

and then hit the carriage return. From then on he can execute the above sequence by simply typing "AKEY". Now, suppose he wants to run ten root-loci in a row, each with a different pole location. He can type:

<u>CREATE</u>, <u>AKEY</u>, <u>CALC</u>, <u>OLPOLE(3)=X</u>, <u>AKEY</u>

Now when he types "AKEY", TOTAL will pause in calculator mode to give the user a chance to enter a number into the X register.

After the user types "C" to leave calculator mode, execution continues and the pole location is changed to the number he entered. The final command in the string, "AKEY", allows AKEY to call itself and the sequence repeats, stopping in CALCULATOR mode for the next pole location. This process continues until the user types a $ to abort the endless loop. <u>The user is cautioned that if he does not include a pause to request input some place in the loop, he will not have a chance to end the loop with a "$" and will have to abort the program completely with a "%A"</u>.

Since each macro key can have 50 steps, the user can write a 150 step program by having one key call another.

Macro keys are particularly useful for reducing large complicated block diagrams at a single command. A user can store each block of the diagram into one of TOTAL's 28 transfer function registers and then write a key using the COPY command and options 21, 22, 23, and 24 to reduce the diagram to a single transfer function for analysis. The user can then modify any variable in any block and instantly reduce it to a single function with one key.

Other uses for TOTAL's three macro keys should be apparent to the user.

SECTION 4. TOTAL'S VARIABLES

TOTAL's data base is divided into variables of four types: scalar constants, polynomial arrays of up to 51 coefficients, root arrays of up to 50 complex numbers, and matrices of up to 10 x 10 elements. Each of these variables may be listed by typing its variable name. Variable values may be assigned or modified as described in the following sections:

4.1 SCALAR VARIABLES

A list of scalar variable definitions is given at the end of this section. The user can modify the values of these variables by simply setting the variable equal to a number or another variable. The following list shows some of the possible ways to modify GAIN, a typical scalar variable:

```
GAIN= 27.98       --   GAIN is set equal to 27.98
GAIN= OLK         --   GAIN is set equal to the current value
                       of the scalar variable OLK
GAIN= POLYA(3)    --   GAIN is set equal to the third coefficient
                       (from the highest) of POLYA
GAIN= GZERO(1,2)--     GAIN is set equal to the imaginary part
                       of the first GTF zero
GAIN= OLPOLE(3) --     GAIN is set to the real part (by default)
                       of the third OLTF pole
GAIN= AMAT(3,4) --     GAIN is set equal to the value of the
                       element in the third row and fourth
                       column of AMAT.
```

4.2 POLYNOMIAL ARRAYS

Polynomial coefficients are stored in arrays from highest to lowest terms. For example, a second-order polynomial stored in POLYA would have the form

POLYA(1)s^2 + POLYA(2)s + POLYA(3)

The user may list any coefficient of this polynomial by typing its name "POLYA(2)" etc. Typing "POLYA" without subscripts will list all coefficients.

The user can modify any coefficient by setting it equal to its desired value or another variable. The following list shows some of the ways to modify the second coefficient of POLYA (POLYA(2)). For example:

```
POLYA(2)=3.38                      POLYA(2)=HPOLE(2,1)
POLYA(2)=X                         POLYA(2)=HPOLE(2)
POLYA(2)=CLNPOLY(3)                POLYA(2)=DMAT(7,3)
```

A list of TOTAL's twelve polynomials and their definitions is given at the end of this section. See the introduction to Section 2.7 for further information on the use of these variables.

4.3 ROOT ARRAYS

For every polynomial array there is a corresponding root array which contains a real and an imaginary part. The Ith root in the ROOTA array, for example, has the form

ROOTA(I,1) + jROOTA(I,2)

where the second subscript designates the real or imaginary part of the root's location in the complex plane. Roots may also be referred to without the second subscript, in which case the real part is assumed. Typing "ROOTA" will list all the roots in the array while typing "ROOTA(2)" will list only the real and imaginary parts of the second root.

The real part of the second root in the ROOTA may be modified in any of the following ways.

```
ROOTA(2,1)= -2.5                 ROOTA(2)= CLPOLE(6,1)
ROOTA(2)=   -2.5               ROOTA(2,1)= GAIN
ROOTA(2)=   POLYA(3)             ROOTA(2)= KMAT(3,2)
```

Similarly, the imaginary part of the fourth open-loop zero would be changed like this:

```
OLZERO(4,2)= 32.8
OLZERO(4,2)= CLPOLE(3,2)
OLZERO(4,2)= GNPOLY(13)
```

It is possible to change both real and imaginary parts of a root at the same time as follows:

```
HPOLE(3)= CLPOLE(2) -- Sets real and imaginary parts of
                       the third HTF pole equal to the
                       real and imaginary parts of the
                       second CLTF pole respectively
ROOTB(2)= GZERO(9)  -- Sets real and imaginary parts of
                       the second root of POLYA equal to
                       the ninth zero of GTF
CLPOLE(4)= 2.78;9   -- (See paragraph below)
```

The reader's attention is called to the last example where a semi-colon was used to separate the desired real and imaginary values of CLPOLE(4). This is a special notation which tells TOTAL to look for another number and assign it to the next variable in the array. Since CLPOLE(4,1) was set equal to -2.78, the semi-colon tells TOTAL to set CLPOLE(4,2) equal to 9. Without the semi-colon, TOTAL would assume the 9 was an option number and go execute option 9! A list of TOTAL's twelve root arrays is included in the variable definitions at the end of this section. See the introduction to Section 2.7 for further information on the use of these variables.

4.4 MATRIX ARRAYS

TOTAL has seven working matrix arrays of maximum dimension 10 x 10 which may be manipulated directly. (The 19 auxiliary matrices provided with the COPY command (see Section 3) can only be modified by first transferring their contents into one of the seven working matrices described below.) These arrays are defined at the end of this section.

The user may list any element of a working matrix by typing its name and indicies. For example, the element in the third row and second column of AMAT is listed by typing "AMAT(3,2)". The entire row can be listed by typing AMAT(3). To list a column, the user must use a prefix: "COL,AMAT(2)". Typing "AMAT" without subscripts will list the entire matrix. (The 19 auxiliary matrices may also be listed in this manner, but subscripts are not allowed.)

To specify the value of a particular element, say AMAT(3,2), the user may type:

```
AMAT(3,2)= 77
AMAT(3,2)= X
AMAT(3,2)= POLYA(11)
```

and so on. The special semi-colon feature described in Section 4.3 may be used to increment the column index by one. For example, typing:

```
AMAT(3,2)= 14;28;64
```

would set AMAT(3,2)= 14, AMAT(3,3)= 28 and AMAT(3,4)= 64. If the maximum row dimension is reached during such a string, TOTAL will move to the first column of the next row, and so on.

4.5 TRANSFER FUNCTIONS

Transfer functions in TOTAL are not separate storage locations, they are just groups of other variables as defined below:

```
 GTF = GNPOLY/GDPOLY = (GNK*GZERO)/(GDK*GPOLE)
 HTF = HNPOLY/HDPOLY = (HNK*HZERO)/(HDK*HPOLE)
OLTF = OLNPOLY/OLDPOLY = (OLNK*OLZERO)/(OLDK*OLPOLE)
CLTF = CLNPOLY/CLDPOLY = (CLNK*CLZERO)/(CLDK*CLPOLE)
```

The variables which make up each of these transfer functions may be listed or modified individually as described in Sections 4.1, 4.2, and 4.3. The contents of every variable associated with a particular transfer function may be listed by typing a corresponding transfer function name. The contents of all variables in one transfer function may be transferred to the respective variables of another using the COPY command described in Section 3.

The four transfer functions described above are called "working" transfer functions because their contents may be manipulated directly. There are also 24 auxiliary transfer functions which may be used only for storage and which cannot be manipulated directly. These auxiliary storage locations are accessable only with the COPY command as described in Section 3.

SCALAR VARIABLE DEFINITIONS

PAK	= POLYA(1)	Polynomial A constant
PBK	= POLYB(1)	Polynomial B constant
PCK	= POLYC(1)	Polynomial C constant
PDK	= POLYD(1)	Polynomial D constant
GNK	= GNPOLY(1)	GTF numerator constant
GDK	= GDPOLY(1)	GTF denominator constant
HNK	= HNPOLY(1)	HTF numerator constant
HDK	= HDPOLY(1)	HTF denominator constant
OLNK	= OLNPOLY(1)	OLTF numerator constant
OLDK	= OLDPOLY(1)	OLTF denominator constant
CLNK	= CLNPOLY(1)	CLTF numerator constant
CLDK	= CLDPOLY(1)	CLTF denominator constant

GK	= GNK/GDK
HK	= HNK/HDK
OLK	= GAIN * (OLNK/OLDK) = GAIN * GK * HK
CLK	= CLNK/CLDK

GAIN	= Added gain in the open loop
TSAMP	= Sampling time in seconds
FACTOR	= Scale factor for calcomp plot size
X,Y,Z,T	= Calculator stack registers
REGISTER(I)	= Subscripted name for referencing the 20 scalar calculator memory registers

The remaining scalars are special purpose root locus variables and are described in greater detail in Section 2.5.

AA	= Right root locus boundary
BB	= Top root locus boundary
CC	= Left root locus boundary
DD	= Bottom root locus boundary
BOUND	= Boundary scale factor
ZETA	= Damping ratio of interest
RAD	= Distance from origin in s-plane or z=1 in z-plane along the zeta of interest at which the search for a locus intersection starts. (actually starting ω_n in both planes)
GTOL	= Range of interest around GAIN
DEL	= Calculation step size between locus points
DELPR	= Printing step size between locus points
FIGURE	= Optional figure number on locus plots

POLYNOMIAL DEFINITIONS

POLYA	Coefficients of polynomial A
POLYB	Coefficients of polynomial B
POLYC	Coefficients of polynomial C
POLYD	Coefficients of polynomial D
GNPOLY	GTF numerator polynomial coefficients
GDPOLY	GTF denominator polynomial coefficients
HNPOLY	HTF numerator polynomial coefficients
HDPOLY	HTF denominator polynomial coefficients
OLNPOLY	OLTF numerator polynomial coefficients
OLDPOLY	OLTF denominator polynomial coefficients
CLNPOLY	CLTF numerator polynomial coefficients
CLDPOLY	CLTF denominator polynomial coefficients

ROOT ARRAY DEFINITIONS

ROOTA	Roots of POLYA
ROOTB	Roots of POLYB
ROOTC	Roots of POLYC
ROOTD	Roots of POLYD
GZERO	GTF zeros = roots of GNPOLY
GPOLE	GTF poles = roots of GDPOLY
HZERO	HTF zeros = roots of HNPOLY
HPOLE	HTF poles = roots of HDPOLY
OLZERO	OLTF zeros = roots of OLNPOLY
OLPOLE	OLTF poles = roots of OLDPOLY
CLZERO	CLTF zeros = roots of CLNPOLY
CLPOLE	CLTF poles = roots of CLDPOLY

MATRIX DEFINITIONS

AMAT	Continuous System Matrix
BMAT	Continuous Input Distribution Matrix
CMAT	Output Matrix
DMAT	Direct Transmission Matrix
KMAT	State-variable Feedback Matrix
FMAT	Discrete System Matrix
GMAT	Discrete Input Distribution Matrix

TRANSFER FUNCTION DEFINITIONS

```
GTF   =  GNPOLY/GDPOLY  =    (GNK*GZERO)/(GDK*GPOLE)
HTF   =  HNPOLY/HDPOLY  =    (HNK*HZERO)/(HDK*HPOLE)
OLTF  = OLNPOLY/OLDPOLY =  (OLNK*OLZERO)/(OLDK*OLPOLE)
CLTF  = CLNPOLY/CLDPOLY =  (CLNK*CLZERO)/(CLDK*CLPOLE)
```

SECTION 5. TOTAL'S SCIENTIFIC CALCULATOR

TOTAL's calculator uses reverse polish notation and is modeled after an HP-45 hand calculator. It has a stack of four registers, X, Y, Z, and T, where X is the display or working register, and twenty memory registers. The user may enter CALCULATOR mode at any time by typing a "C" and leaves CALCULATOR mode the same way. CALCULATOR mode is designated by the prompt **, and the user may obtain a listing of calculator keys by typing "KEYS" as shown below.

```
** KEYS

ROLL            CHS         LOG           MEMORY
EXCHANGE        SIN         ALOG          STORE
CLX             COS         POLAR         RECALL
CLEAR           TAN         RECTANG       LASTX
RECIPROCAL      ASIN        FIX           DTOR
SQUARE          ACOS        SCI           RTOD
SQROOT          ATAN        LIST          DEGREES
YTOX            LN          STACK         RADIANS
PI              EXP         FASTSTACK     KEYS
```

The meaning of these keys will be discussed later in this section.

To enter a number into the X register, the user simply types it. Numbers are entered automatically when followed by a blank, comma, or carriage return. Typing any of the variable names listed in Section 4 automatically enters the variable's value into the X register. (Typing a complex variable name with only one subscript will put the real part into X and the imaginary part into Y.)

An example of entering the numbers 1.0, 2.0, 3.0 and 4.0 is shown on the following page:

```
X=                                0.000000

** 1 2 3 4 STACK

T=                                1.000000
Z=                                2.000000
Y=                                3.000000
X=                                4.000000
```

The 1 is initially entered into the X register until it is pushed up into Y by the 2. Subsequently, the 3 pushes the 2 and 1 up another notch. Finally, the 4 is entered leaving the contents of the four stack registers as shown by the STACK command.

Once numbers have been entered, all operations occur between X and Y. For example, typing a "+" would add X to Y and leave the sum in X. The contents of Z then drop into Y and T drops into Z. The contents of register T are unaffected. Similar results are obtained by typing "-", "*", or "/", where "/" divides Y by X and "-" subtracts X from Y.

One final example should clarify stack operation. If the user wants to evaluate

(3.14 * 2.78 + 8.62 * 98.6)/6.02

he would type

```
** 3.14 2.78 *  8.62 98.6 * +  6.02 /
```

and TOTAL would print out the answer. What actually occurs in the stack is this:

1. 3.14 is entered into X.
2. 2.78 is entered into X shoving 3.14 into Y.
3. X and Y are multiplied leaving 8.73 in X.
4. 8.62 is entered into X shoving 8.73 into Y.
5. 98.6 is entered shoving 8.62 into Y and 8.73 into Z.
6. X and Y are multiplied leaving 849.93 in X and dropping 8.73 back into Y.

7. X and Y are added leaving 858.66 in X.
8. 6.02 is entered into X shoving 858.66 into Y.
9. Y is divided by X leaving 142.63 in X.
10. TOTAL prints out the contents of the X register as the answer.

For further information on stack operations the user is referred to someone who owns a Hewlett-Packard calculator.

All that remains now is to define the meaning of each key. In the descriptions below, "old" refers to status before the key is executed and "new" refers to status afterwards. Underlined letters in each command show minimum abbreviation allowed.

ROLL	Moves old Y to new X, old Z to new Y, old T to new Z and old X to new T
EXCHANGE	Moves old X to new Y and old Y to new X
CLX	Clear X (sets X to 0.0)
CLEAR	Clear stack (sets X, Y, Z, and T to 0.0)
RECIPROCAL	Sets new X equal to reciprocal of old X
SQUARE	Sets new X equal to old X squared
SQROOT	Sets new X equal to square root of old X
YTOX	Sets new X equal to old Y to the old X power and move old Z into new Y and old T into new Z.
PI	Enters 3.14159265358979 into new X, old X into new Y, old Y into new Z and old Z into new T.
CHS	Change sign of X
SIN	Sets new X equal to sin(old X)
COS	Sets new X equal to cos(old X)
TAN	Sets new X equal to tan(old X)
ASIN	Sets new X equal to arcsin(old X)
ACOS	Sets new X equal to arccos(old X)
ATAN	Sets new X equal to arctan(old X)
LN	Sets new X equal to natural log(old X)
LOG	Sets new X equal to common log(old X)
EXP	Sets new X equal to exp(old X)
ALOG	Sets new X equal to $10^{(\text{old } X)}$
POLAR	Converts X and Y in rectangular coordinates to polar, where X is the magnitude and Y is the angle

RECTANG	Converts X magnitude and Y angle to X and Y rectangular coordinates
FIX	FIX followed by any number from 0 to 14 sets all future calculator outputs to that many decimal places
SCI	Similar to FIX except that numbers are thereafter expressed in scientific notation with the specified number of decimal places
LIST	Prints contents of X register. Useful for displaying intermediate results in a long string of calculator commands
STACK	Prints contents of X, Y, Z, and T vertically
FASTSTACK	Prints contents of X, Y, Z, and T horizontally
MEMORY	Prints contents of all 20 memory registers
STORE	Must be followed by a number from 1 to 20. Stores contents of X to specified memory register
RECALL	Must be followed by a number from 1 to 20. Recalls the contents of specified memory register to X and shoves old X to new Y, old Y to new Z and old Z to new T
RCL	Another abbreviation for RECALL
LASTX	Recalls the value of X prior to the last operation
DTOR	Converts old X in degrees to new X in radians
RTOD	Converts old X in radians to new X in degrees
DEGREES	Puts calculator into degree mode and turns the switch DEGREE on
RADIANS	Puts calculator into radian mode and turns the switch DEGREE off
KEYS	Types out a list of all calculator keys

The user may use the calculator at any time -- even in the middle of an option during input of data. For further information see Section 1.1.

SUMMARY OF TOTAL'S OPTIONS

```
COMMAND- ATTACH,TOTAL,ID=AFIT

COMMAND- TOTAL

WELCOME TO TOTAL--TYPE HELP FOR INTRO--TYPE STOP TO STOP

OPTION >   HELP

TOTAL IS AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM
FOR DIGITAL & CONTINUOUS CONTROL SYSTEM ANALYSIS.
IT CONTAINS 100 OPTIONS DIVIDED INTO GROUPS OF 10
ACCORDING TO GENERAL APPLICATION.

OPTIONS ENDING IN 0 LIST THE NEXT 10 OPTIONS,
FOR EXAMPLE, OPTION 30 LISTS OPTIONS 30 THRU 39.

THE FOLLOWING ARE THE MAIN OPTION GROUPS:

  0-9:  TRANSFER FUNCTION INPUT OPTIONS
10-19:  MATRIX INPUT OPTIONS
20-29:  BLOCK DIAGRAM MANIPULATION OPTIONS
30-39:  TIME RESPONSE OPTIONS
40-49:  ROOT LOCUS OPTIONS
50-59:  FREQUENCY RESPONSE OPTIONS
60-69:  POLYNOMIAL OPTIONS
70-79:  MATRIX OPERATIONS
80-89:  DIGITIZATION OPTIONS
90-99:  MISCELLANEOUS OPTIONS

WHEN INPUT IS REQUESTED BY TOTAL, THE USER MAY:

ENTER THE REQUESTED INFORMATION, OR:
TYPE  ?  FOR EXPLANATION OF INPUT NEEDED,
TYPE  L  FOR A LIST OF CURRENT VARIABLE VALUES,
TYPE  *  TO LEAVE AN ITEM UNCHANGED,
TYPE  C  TO USE CALCULATOR BEFORE OR DURING INPUT,
TYPE  $  TO ABORT THE OPTION,
TYPE  X  OR Y, Z, T, OR R1 THRU R20 TO TELL TOTAL
         TO GET REQUESTED INFO FROM CORRESPONDING
         CALCULATOR REGISTER.

TO END TOTAL, TYPE  STOP.
```

TRANSFER FUNCTION INPUT OPTIONS

```
*  0    LIST OPTIONS
*  1    RECOVER ALL DATA FROM FILE MEMORY
*  2    POLYNOMIAL FORM--GTF
*  3    POLYNOMIAL FORM--HTF
*  4    POLYNOMIAL FORM--OLTF
*  5    POLYNOMIAL FORM--CLTF
*  6    FACTORED FORM----GTF
*  7    FACTORED FORM----HTF
*  8    FACTORED FORM----OLTF
*  9    FACTORED FORM----CLTF
```

MATRIX INPUT OPTIONS

```
* 10    LIST OPTIONS
* 11    AMAT--CONTINUOUS SYSTEM MATRIX
* 12    BMAT--CONTINUOUS INPUT MATRIX
* 13    CMAT--OUTPUT MATRIX
* 14    DMAT--DIRECT TRANSMISSION MATRIX
* 15    KMAT--STATE VARIABLE FEEDBACK MATRIX
* 16    FMAT--DISCRETE SYSTEM MATRIX
* 17    GMAT--DISCRETE INPUT MATRIX
* 18    SET UP STATE SPACE MODEL OF SYSTEM
* 19    EXPLAIN USE OF ABOVE MATRICES
```

BLOCK DIAGRAM MANIPULATION AND STATE SPACE OPTIONS

```
* 20    LIST OPTIONS
* 21    FORM OLTF = GTF * HTF   (IN CASCADE)
* 22    FORM CLTF = (GAIN*GTF)/ (1 + GAIN*GTF*HTF)
* 23    FORM CLTF = (GAIN*OLTF)/(1 + GAIN*OLTF)
* 24    FORM CLTF = GTF + HTF   (IN PARALLEL)
* 25    GTF(s) AND HTF(s) FROM CONTINUOUS STATE SPACE MODEL
* 26    GTF(z) AND HTF(z) FROM DISCRETE STATE SPACE MODEL
* 27    WRITE ADJOINT(sI - AMAT) TO FILE ANSWER
* 28    FIND HTF FROM CLTF & GTF FOR CLTF = GTF*HTF/(1+GTF*HTF)
* 29    FIND HTF FROM CLTF & GTF FOR CLTF =   GTF  /(1+GTF*HTF)
```

TIME RESPONSE OPTIONS

```
* 30    LIST OPTIONS
* 31    TABULAR LISTING OF F(t) OR F(k)
* 32    PLOT F(t) OR F(k) AT USER'S TERMINAL
* 33    PRINTER PLOT (WRITTEN TO FILE ANSWER)
* 34    CALCOMP PLOT (WRITTEN TO FILE  PLOT )
* 35    PRINT TIME OR DIFFERENCE EQUATION (F(t) OR F(k))
* 36    PARTIAL FRACTION EXPANSION OF CLTF (OR OLTF)
* 37    LIST T-PEAK, T-RISE, T-SETL, T-DUP, M-PEAK, FINAL VALUE
* 38    QUICK SKETCH AT USER'S TERMINAL
* 39    SELECT INPUT: STEP, RAMP, PULSE, IMPULSE, SINE
```

ROOT LOCUS OPTIONS

* 40	LIST OPTIONS
* 41	GENERAL ROOT LOCUS
* 42	ROOT LOCUS WITH A GAIN OF INTEREST
* 43	ROOT LOCUS WITH A ZETA (DAMPING) OF INTEREST
* 44	LIST N POINTS ON A BRANCH OF INTEREST
* 45	LIST ALL POINTS ON A BRANCH OF INTEREST
* 46	LIST LOCUS ROOTS AT A GAIN OF INTEREST
* 47	LIST LOCUS ROOTS AT A ZETA OF INTEREST
* 48	PLOT ROOT LOCUS AT USER'S TERMINAL
* 49	LIST CURRENT VALUES OF ALL ROOT LOCUS VARIABLES

FREQUENCY RESPONSE OPTIONS

* 50	LIST OPTIONS
* 51	TABULAR LISTING
* 52	TWO CYCLE SCAN OF MAGNITUDE
* 53	TWO CYCLE SCAN OF PHASE
* 54	PLOT F(w) AT USER'S TERMINAL
* 55	CALCOMP PLOT--LINEAR FREQUENCY AXIS
* 56	CALCOMP PLOT--LOG FREQUENCY AXIS
* 57	TABULATE POINTS OF INTEREST: PEAKS, BREAKS, ETC.
* 58	CALCOMP PLOT--NYQUIST POLAR PLOT
* 59	CALCOMP PLOT--NICHOLS LOG-MAG/ANGLE PLOT

POLYNOMIAL OPERATIONS

* 60	LIST OPTIONS	
* 61	FACTOR POLYNOMIAL (POLYA)	
* 62	ADD POLYNOMIALS	(POLYC = POLYA + POLYB)
* 63	SUB POLYNOMIALS	(POLYC = POLYA - POLYB)
* 64	MULTIPLY POLYS	(POLYC = POLYA * POLYB)
* 65	DIVIDE POLYS	(POLYC + REM = POLYA / POLYB)
* 66	STORE POLY_ INTO POLYD	
* 67	EXPAND ROOTS INTO A POLYNOMIAL	
* 68	(s + a)**^{n} EXPANSION INTO A POLYNOMIAL	
* 69	ACTIVATE POLYNOMIAL CALCULATOR	

MATRIX OPERATIONS

* 70	LIST OPTIONS
* 71	ROOTA = EIGENVALUES OF AMAT
* 72	CMAT = AMAT + BMAT
* 73	CMAT = AMAT - BMAT
* 74	CMAT = AMAT * BMAT
* 75	CMAT = AMAT INVERSE
* 76	CMAT = AMAT TRANSPOSED
* 77	CMAT = IDENITY MATRIX
* 78	DMAT = ZERO MATRIX
* 79	COPY ONE MATRIX TO ANOTHER

DIGITIZATION OPTIONS

```
* 80    LIST OPTIONS
* 81    CLTF(s) TO CLTF(z) BY IMPULSE INVARIANCE
* 82    CLTF(s) TO CLTF(z) BY FIRST DIFFERENCE APPROXIMATION
* 83    CLTF(s) TO CLTF(z) BY TUSTIN TRANSFORMATION
* 84    CLTF(z) TO CLTF(s) BY IMPULSE INVARIANCE
* 85    CLTF(z) TO CLTF(s) BY INVERSE FIRST DIFFERENCE
* 86    CLTF(z) TO CLTF(s) BY INVERSE TUSTIN
* 87    FIND FMAT AND GMAT FROM AMAT AND BMAT
* 88    COMPUTE FMAT = exp(AMAT * T)
* 89    CLTF(X) TO CLTF(Y) BY  X = ALPHA*(Y + A)/(Y + B)
```

MISCELLANEOUS OPTIONS

```
* 90    LIST OPTIONS
* 91    UPDATE MEMORY FILE WITH CURRENT DATA
  92
* 93    LIST CURRENT SWITCH SETTINGS (ECHO, ANSWER, ETC)
  94
  95
* 96    LIST SPECIAL COMMANDS ALLOWED IN OPTION MODE
* 97    LIST VARIABLE NAME DIRECTORY
* 98    LIST MAIN OPTIONS OF TOTAL
* 99    GIVE THE INTRODUCTION TO TOTAL
```

APPENDIX B

PROGRAMMER'S MANUAL FOR TOTAL

(MARCH 1978)

Contents

Appendix B

SECTION 1. INTRODUCTION

This manual is written to provide documentation on all programs and subprograms used in TOTAL as well as a description of the overall structure and internal operation of the program. It is intended for the individual who wishes to make his own modification and additions to the coding and as a general aid in maintaining the program or transporting it from one computer system to another.

The reader is assumed to be thoroughly familiar with the external operation of the program. It is recommended that he read the User's Manual for TOTAL completely before beginning to use this manual.

SECTION 2. DESCRIPTION OF OVERALL STRUCTURE

TOTAL is a large program. In its absolute form, with all external references satisfied, it requires in eccess of $600,000_8$ words of central memory for execution. Since many computer systems do not have this much memory available, and most limit interactive users to a much smaller amount (on the order of 60K), TOTAL has been designed with a structure of overlays so as to not require more than 60K at any one time. The following is a brief description of that structure:

2.1 HOW OVERLAYS WORK

Overlays are simply a way of dividing a large program into a series of smaller programs, each of which will fit into the available amount of core memory. As each of these programs (overlays) is needed, it is loaded into memory replacing one which has just finished executing. A small executive routine, written to control the overall flow of the program, is responsible for calling each overlay into memory as it is needed. This executive is called the main overlay and remains in core memory at all times. The program segments which it controls are called primary overlays. Only one primary overlay may be in central memory with the main overlay at a time. It is also possible to have another level of overlays which can be called by a given primary overlay. These programs are called secondary overlays.

Each overlay is a separate program which is loaded into memory and executed when it is called by a higher level overlay.

Information is passed between overlays through labeled common statements in the main overlay.

The use of overlays should become more apparent when their use in TOTAL is described in the next section. For further information, the reader is referred to the CDC FORTRAN Reference Manual and the CDC Loader Reference Manual.

2.2 TOTAL'S OVERLAY STRUCTURE

TOTAL in its present form uses one main overlay, seventeen primary overlays, and eleven secondary overlays.

The main overlay simply holds the common data arrays, establishes their default values, and calls each of the primary overlays when needed, as determined by a short decision-making routine. This overlay is described in detail in Section 3.

The primary overlays perform all of the actual operations in TOTAL. Each is responsible for a certain class of functions which may include option and command execution, variable definition and modification, switch setting, user assistance, and interactive user interfacing. If an overlay is too large for the given core restriction, it is divided into secondary overlays that will fit. Primary and secondary overlays are discussed in detail in Section 4.

Each overlay program is allowed any number of subprograms, each of which takes up some of the memory space allowed for that overlay. These subroutines, are described in depth in Section 5.

SECTION 3. DESCRIPTION OF MAIN EXECUTIVE OVERLAY

TOTAL's main executive overlay performs four functions vital to program operation. First, it houses in its PROGRAM statement the definition of all input-output devices throughout the program. Second, it contains all of the labeled common statements for the data base (variables which are used by more than one program and which must remain in central memory at all times). Third, it initializes the values of all the variables in the common data base. Finally, it contains the routine which selects other overlays and controls the flow of the entire program. Since a thorough understanding of this overlay is essential to further modifications and additions, the remainder of this section is devoted to a complete description of its structure.

3.1 THE PROGRAM STATEMENT

The program statement for TOTAL is shown below.

```
 PROGRAM TOTAL(INPUT=100B,OUTPUT=100B,AKEY=100B,BKEY=100B,
+CKEY=100B,ANSWER,MEMORY,DOODLE=100B,PLOT,TAPE1=AKEY,
+TAPE2=BKEY,TAPE3=CKEY,TAPE5=INPUT,TAPE6=ANSWER,
+TAPE7=OUTPUT,MEMAUX=1000B,TAPE8=MEMAUX,TAPE9=MEMORY,
+TAPE10=DOODLE)
```

It performs three functions, including definition of local file name, reference number, and buffer length for each I/O device (tape) used by TOTAL. The name and purpose of each of these tapes is given in Table I.

The choice of buffer size for each of these tapes is a trade-off between amount of memory used and speed of execution.

TABLE I

Input/Output "Files" used by TOTAL

Tape number	Tape name	Buffer size	Purpose
TAPE 1	AKEY	100B	Storage for AKEY macro
TAPE 2	BKEY	100B	Storage for BKEY macro
TAPE 3	CKEY	100B	Storage for CKEY macro
TAPE 4	PLOT	2000B	Calcomp plot output
TAPE 5	INPUT	100B	Connected to user's terminal for input
TAPE 6	ANSWER	2000B	Auxiliary output tape
TAPE 7	OUTPUT	100B	Connected to user's terminal for output
TAPE 8	MEMAUX	1000B	Auxiliary memory mass storage tape
TAPE 9	MEMORY	2000B	Memory storage tape
TAPE 10	DOODLE	100B	Scratch tape for internal use only

A buffer is just a location in central memory where output is placed until it can be written to a tape by an output device. Since output is usually generated much faster than it can be written to tape, the buffer often fills up and execution must be interrupted until the output device can catch up. This slows down execution time. By increasing the buffer size, fewer interruptions are needed and the program runs faster at the expense of space in central memory. Since space in a program of this size is at a premium, buffer size must be kept at the minimum value consistant with reasonably fast operation.

The rule of thumb used in selecting buffer sizes in TOTAL was that frequently used files which receive output in large volumes are given large buffers (1000B to 2000B) while all others are kept at the minimum size of 100B = 100_8 = 77_{10} words of central memory. Even with this careful assignment, 7600_8 words of central memory out of the allowed $60{,}000_8$ are devoted to buffer use in TOTAL. Should more space ever be needed for any reason, reducing this buffer size, at the expense of execution time, is a possible solution.

3.2 TOTAL'S COMMON DATA BASE

Figure 1 shows the labeled common statements which make up TOTAL's data base. Every variable which must be passed from overlay to overlay or which is needed throughout the program for any reason must be placed in this array. Numbers not located in common are lost each time a new overlay replaces an old one.

```
      COMMON/TOTAL1/AMAT(10,10),NA,MA,BMAT(10,10),NB,MB,CMAT(10,10),
     +NC,MC,DMAT(10,10),ND,MD,FMAT(10,10),NF,MF,GMAT(10,10),NG,MG,
     +AKMAT(10,10),NK,MK
      COMMON/TOTAL7/POLYA(51),POLYB(51),POLYC(51),POLYD(51),
     +ROOTA(50,2),ROOTB(50,2),ROOTC(50,2),ROOTD(50,2),
     +NPA,NPB,NPC,NPD,PAK,PBK,PCK,PDK
      COMMON/TOTAL9/GNPOLY(51),GDPOLY(51),GZERO(50,2),GPOLE(50,2),
     +NGZ,NGP,GK,GNK,GDK
      COMMON/TOTL10/HNPOLY(51),HDPOLY(51),HZERO(50,2),HPOLE(50,2),
     +NHZ,NHP,HK,HNK,HDK
      COMMON/TOTL11/OLNPOLY(51),OLDPOLY(51),OLZERO(50,2),OLPOLE(50,2),
     +NOLZ,NOLP,OLK,OLNK,OLDK
      COMMON/TOTL12/CLNPOLY(51),CLDPOLY(51),CLZERO(50,2),CLPOLE(50,2),
     +NCLZ,NCLP,CLK,CLNK,CLDK
      COMMON/TOTL14/NOPT,IOPT,JOPT,KOPT,LOPT,MOPT,LPROMT,LASTOPT,NGO
      COMMON/TOTL15/IFLAG(100),JFLAG(100),KFLAG(20),LFLAG(20),NCALL(20)
      COMMON/TOTL16/X,Y,Z,T,REG(20),XLAST
      COMMON/TOTL17/MCOMM(100),DATM(100),MPT
      COMMON/TOTL18/NRPT,NROUTE(10),NPT
      COMMON/PARTL1/ZZZ(50),XR,NXX,YYY(50),W(50),CLOSE
      COMMON/PARTL2/CR(50),EC(50),OM(50),FE(50),LL,DATIN(4)
      COMMON/PARTL4/FAX(2),AXMIN,FRANGE,FTMAX,FTMIN
      COMMON/PARTL5/RSLOPE,RWIDTH,RHIGHT,ROMEGA,INPUTR
      COMMON/ROOTL1/BOUND,FPN,IFOLD,ISTAM,ITITL,ITHES,RAD,ZETA
      COMMON/ROOTL2/LLROOT,DELPR(50),DEL(50),GAIN,GTOL,LN,IPLOT
      COMMON/ROOTL3/XP(50),YP(50),XZ(50),YZ(50),N,M,XSTART,YSTART,TIME
      COMMON/ROOTL4/GA,AA,BB,CC,DD,JK
      COMMON/FREQR1/WMIN,WMAX,DELW,YIN(3),NUP,NDOWN
      COMMON/FREQR2/KIND,CYCLENG,NCYC,IPOW,RANGE,ORDER
      COMMON/FREQR3/AMIN,AMAX,AXLIN(2),AXDB(2),FNMAX,FNMIN
      COMMON/DMULR/POLYQ(51),ROOTQ(50,2),NQ,PQK
      COMMON/READER/LCOM,LODV,LODI,L1DV,LCPLXV,L2DV,LKEY,LABBR,
     +LASTF,LMAT
      COMMON/LOGIC1/REQUEST,ECHO,EXTCALC,CALC,EQUAL,COMMA,LIST
      COMMON/LOGIC2/CLOSED,DECIBEL,HERTZ,DEGREE,ANSWER,PLAT,GRID,SCALE
      COMMON/LOGIC3/TEST,ALREADY
      COMMON/MATRIX1/AIN(10),IRUN,NU,NY,NS,MU,LY
      COMMON/TTYPLT/OK,IPT
      COMMON/PLOTER/MULT,JPLOT,KPLOT,LPLOT,FACT
      COMMON/X153/SCRTCH(153)
      COMMON/DIGIT/TSAMP,SAMPT,ALPHAT,SIGMAA,SIGMAB
      COMMON/MASSTO/INMASS(47)
      DIMENSION ROOTR(50),ROOTI(50)
      LOGICAL REQUEST,ECHO,EXTCALC,CALC,EQUAL,COMMA,LIST,
     +CLOSED,DECIBEL,HERTZ,DEGREE,ANSWER,PLAT,GRID,SCALE
      LOGICAL TEST,ALREADY,GOPLOT
```

Fig. 1. TOTAL's common data base.

Variables in this data base are initialized to their default values at the beginning of the main executive overlay.

3.3 DESCRIPTION OF THE MAIN EXECUTIVE OVERLAY

The discussion in this section is a detailed description of TOTAL's main executive overlay. The reader is referred to the flow chart on the following page which may be helpful in reading the remainder of this section.

When execution of TOTAL begins, all variables in the common data base are initialized as described in Section 3.2. The mass storage file MEMAUX is also opened. TOTAL then enters OPTION mode at statement 11110 and calls program READER (overlay 6) to receive a string of commands, option numbers, etc. from the user. READER compiles the user's input, stores the commands in coded form in the array MCOMM, sets the MPT pointer to the first command in MCOMM and returns control to the executive overlay at statement number 11111. READER is discussed in much greater detail in Section 4.

Statement 11111 is the main control node of the entire program. Every overlay returns control to this point when it has finished executing. From this point, program flow runs through a three level sorter to determine where control should be transfered next. These levels include the internal routing controller, the command type sorter, and the master overlay selector.

Internal routing controller.

This controller is a special utility which allows one overlay to specify a sequence of additional overlays to be called

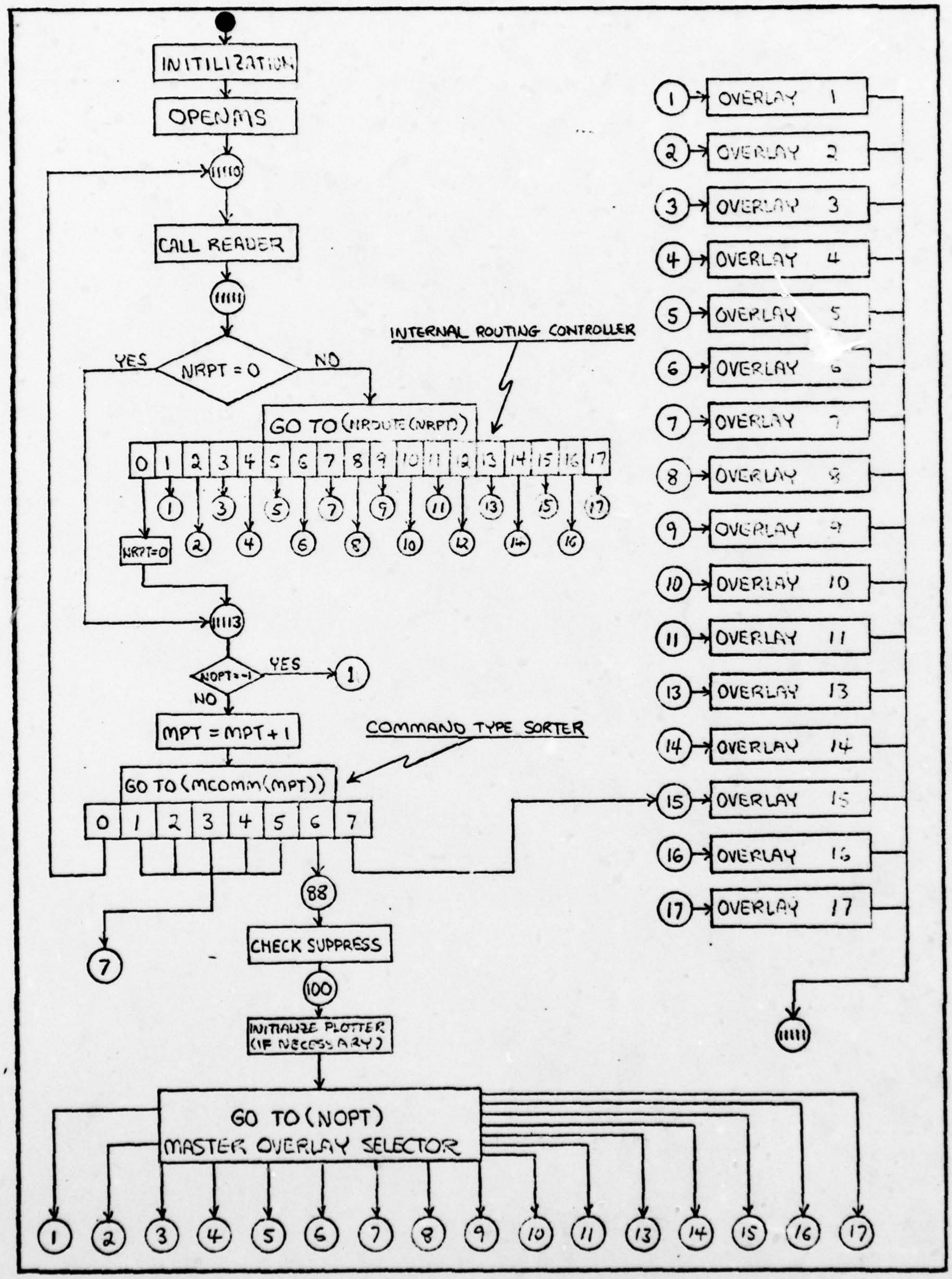


Fig. 2. Flow chart for main executive overlay.

before TOTAL moves on to the next user command. Normally, when the user gives a command, the necessary overlay is selected, it executes the command, and TOTAL moves on to the next command. If, however, for some reason more than one overlay is needed to complete the given command, the first overlay called sets up the sequence of additional overlay calls that will be needed and stores them in the array NROUTE. TOTAL then executes all commands in NROUTE before moving on to the next main command in the MCOMM array.

A typical example of this procedure is in the execution of option 48. Overlay 4 contains all of the necessary routines for calculating points on a root locus. It does not, however, have room for the routine which actually plots the points and an additional overlay (overlay 11) is needed. Now, when overlay 4 has finished computing the points, it stores an 11 in NROUTE(1), sets the NRPT pointer equal to 1, and returns control to the executive. The executive sees that NRPT is not zero and uses the internal routing controller to send control to the overlay number stored in NROUTE(NRPT) = NROUTE(1) = 11. NRPT is then incremented and when overlay 11 has finished, control is transfered to overlay number NROUTE(NRPT) = NROUTE(2) = 0. Whenever a zero is encountered (as in this case), TOTAL knows that the last internal command has been complete. NRPT is set back to zero and program flow passes on to the command type sorter. Since NROUTE has a dimension of 20, up to 20 secondary commands can be specified if needed. An overlay can even have itself called back when the series it specified is finished by simply putting its own overlay number into the NROUTE array. This is how temporary interrupts to CALCULATOR or HELP modes are handled.

Command type sorter.

This routine begins at statement 11113 as shown in Fig. 2. The command type sorter increments the MPT pointer, looks at the coded command stored in MCOMM(MPT), and decides where the program flow should go next. If the next item in the array is a general command or variable operation (MCOMM(MPT) = 1, 2, 3, 4, or 5), program DECODER (overlay 7) is called to decode and execute it. If MCOMM(MPT) = 6, the next command is an option number and control is given to the master overlay selector. The COPY command has an entire program (overlay 15) devoted to it and so, rather than calling DECODER, control is passed to overlay 15 directly. MCOMM(MPT) = 0 indicates that the last command has been executed and the executive returns to statement 11110 to call READER and ask the user for further instructions.

Master overlay selector.

Since each of TOTAL's options may be located, in general, in any one of the 17 overlays, a method is needed to select the correct overlay for a given option. This is done with a massive GO TO statement called the "master overlay selector."

```
101   GO TO( 1, 3, 3, 3, 3, 3, 3, 3, 3, 9,
    +        14,14,14,14,14,14,14,14, 9, 9,
    +        17,17,17,17,14,14,14,17,17, 9,
    +         2, 2, 2, 2, 2, 2, 2, 2, 2, 9,
    +         4, 4, 4, 4, 4, 4, 4, 4, 4, 9,
    +         5, 5, 5, 5, 5, 5, 9, 9, 9, 9,
    +         3, 3, 3, 3, 3, 3, 3, 3, 3, 9,
    +        14,14,14,14,14,14,14,14,14, 9,
    +        16,16,16,16,16,16, 9, 9,16, 9,
    +         1, 9, 13, 9, 18, 9, 9, 9, 9, 9), NOPT
```

Entries in this computed GO TO statement are indexed by the option number, NOPT, and are simply the statement numbers for the corresponding overlay calling statements. For example, if NOPT = 93, control is transferred to statement number 13 which just happens to be the CALL statement for overlay 13, and so on.

The value of NOPT is obtained in statement 88 from an array DATM(MPT) which is indexed by the same pointer, MPT, as the primary command array MCOMM(MPT). When the user types a valid option number, the fact that it is an option number is stored as MCOMM(MPT) = 6 and its actual value stored in DATM(MPT). Complete details on these arrays are given under program READER in Section 4.

<u>Other operations performed in this overlay</u>.

The "CHECK SUPPRESS" block in the flow chart is just a look ahead operation to see if the user has typed an "S" following the option number. If the suppress code <u>is</u> stored in MCOMM(MPT + 1), the logical variable REQUEST is set to .FALSE. to tell TOTAL to skip requesting any input it may need to execute the option and just use the old (or default) values. If the S code is found, MPT is incremented so that the SUPPRESS command itself will not be executed the next time.

If the plotter has not been initialized and the upcoming option will generate a plot, the "INITIALIZE PLOTTER" block will be executed. This routine ensures that the plotter is initialized once and only once. (Every time the plotter is initialized, a new banner is drawn on the PLOT file.)

When the block labeled OPENMS is executed, TOTAL calls the subroutine OPENMS to open the mass storage file MEMAUX. If MEMAUX does not contain any information (from previous runs of TOTAL), this routine fills all 45 storage sections with zeros. If these arrays were not defined in this manner and the user later tried to read from a location to which he had not first written, a fatal error would occur and execution of TOTAL would end abruptly.

After any overlay has finished executing, control is always returned to statement 11111 and the cycles described above are repeated. When the command STOP is finally typed, DECODER sets NOPT = -1 and when statement 11113 is executed, overlay 1 is called to write all data in the common data base to local file MEMORY and stop the program.

SECTION 4. DESCRIPTION OF TOTAL'S OVERLAYS

This section is intended to document the structure, purpose, and general operation of each of TOTAL's primary overlays. The source listings of each of these programs include liberal use of comment statements which are intended to make them self-explanatory to the programmer. This section contains additional information which may be useful in understanding and using the program coding.

4.1 OVERLAY (1,0) -- PROGRAM UPDATE

Program UPDATE is one of TOTAL's smallest overlays. It performs only two functions, including writing information from common to local file MEMORY and reading that information back in.

When option 91 is selected, UPDATE is called by the main executive overlay. Local file MEMORY (TAPE9) is rewound and all variables in the common data base are written onto it. The first number written is the code number "989" used to identify the tape as a MEMORY file. After all information has been saved with a series of simple WRITE statements, control is returned to the main overlay.

The reverse sequence of events takes place when UPDATE executes option 1. MEMORY is rewound and tested to see if its first entry is "989". If it is not, an error message is printed, otherwise, the data is read back into the common data base and the option terminates as before.

When the command STOP is executed, UPDATE does option 91, returns unneeded local files, and terminates TOTAL.

4.2 OVERLAY (2,0) -- PROGRAM TIMER

Overlay (2,0) is just a short executive overlay which calls three secondary overlays as needed to perform digital and continuous time response options. For continuous responses (TSAMP = 0) or partial fraction expansions (NOPT = 36), it calls program PARTL (Overlay (2,1)). For discrete responses (TSAMP ≠ 0) or selecting inputs (NOPT = 39), DIGITR (Overlay (2,2)) is called. If a Calcomp plot is generated (NOPT = 34), PLOTFIN (overlay (2,3)) is called to finish drawing the titles, axes, etc. Sections 4.3, 4.4, and 4.5 describe these three secondary overlays.

4.3 OVERLAY (2,1) -- PROGRAM PARTL

Program PARTL uses a Heaviside partial fraction expansion algorithm taken from the AFIT program of the same name. (Ref.1 Most other features, however, including plots, listings, and function outputs, are new and greatly improved. Fig. 3 provides a general flow chart of the program. The blocks labeled "standard interrupt routine" and "restart" are used to allow temporary interruptions of the program to call the CALCULATOR and HELP overlays. The value of NCALL(4) set and used by these routines simply allows program flow to find its way back to the statement at which the interruption occurred. A complete explanation of these routines is given in Section 5 under subroutine READS.

At the beginning of PARTL, the transfer function of interest

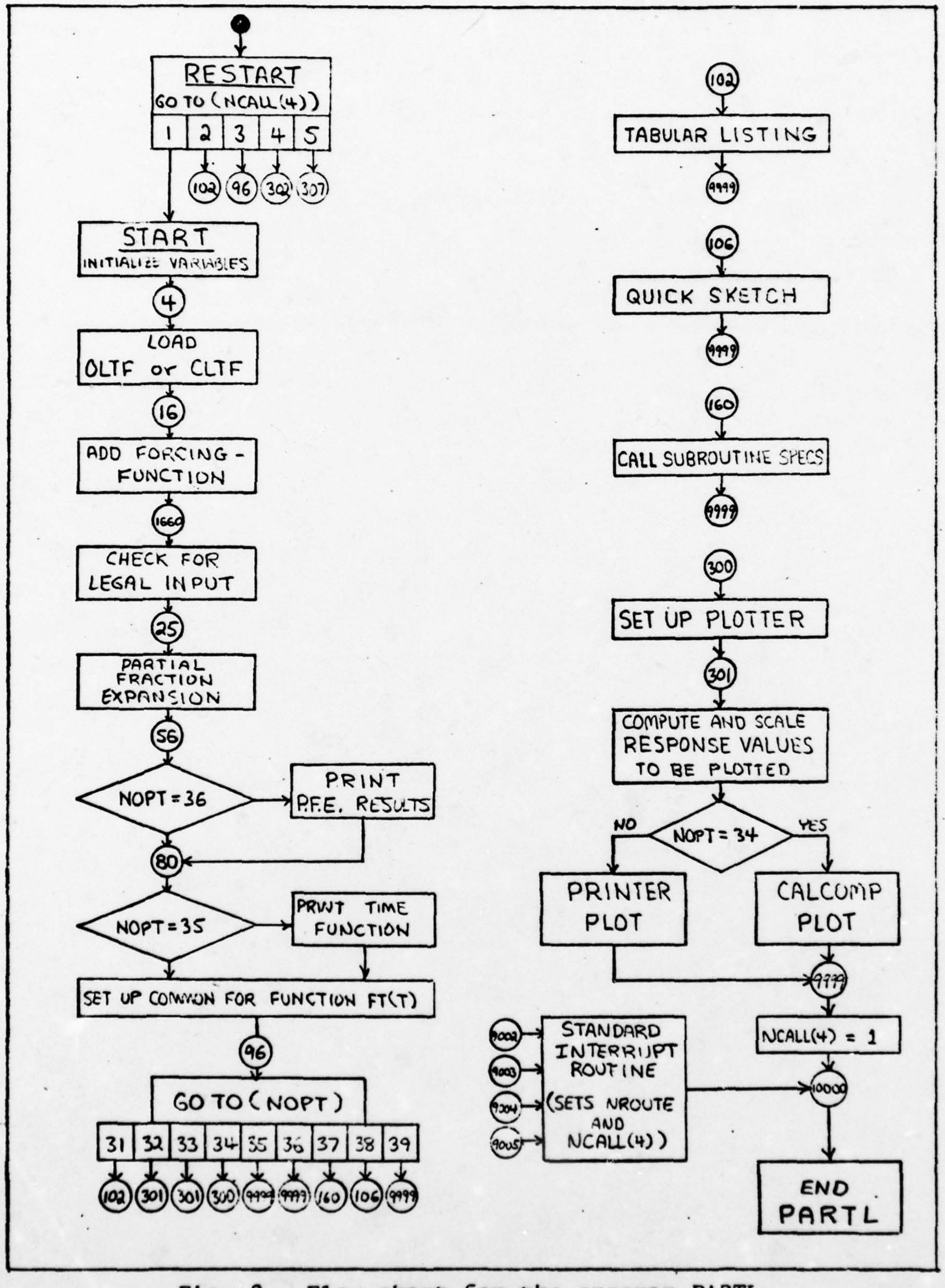


Fig. 3. Flow chart for the program PARTL.

is loaded, multiplied by the transform of the input of interest, and tested for legal form. A partial fraction expansion (PFE) is then performed and the desired time function determined from an inverse Laplace transformation of the PFE terms. Options 35 and 36 print out some of the results of these computations while thay are being obtained. All other options use the general function subprogram FT(T) to obtain the actual time response results. FT(T) gets its information through labeled common statements and is described more fully in Section 5.

4.4 OVERLAY (2,2) -- PROGRAM DIGITR.

Given an nth order transfer function such as

$$\frac{C(z)}{R(z)} = \frac{a_1 z^2 + a_2 z + a_3}{z^2 + b_2 z + b_3}$$

DIGITR calculates a discrete time response by forming the recursive difference equation

$$c(k) = a_1 r(k) + a_2 r(k - 1) + a_3 r(k - 2) - b_2 c(k - 1) - b_3 c(k - 2)$$

and iterating it for some desired input sequence.

Subroutine RIN is used to supply the input sequence (ramp, step, pulse, etc.) and subroutine PROGAT shifts the arrays of past inputs and outputs and evaluates c(k) for each new value of k.

After the response values are calculated over the range of k specified, DIGITR plots and/or tabulates the results in much the same way as PARTL. A flow chart for DIGITR is shown in Fig. 4.

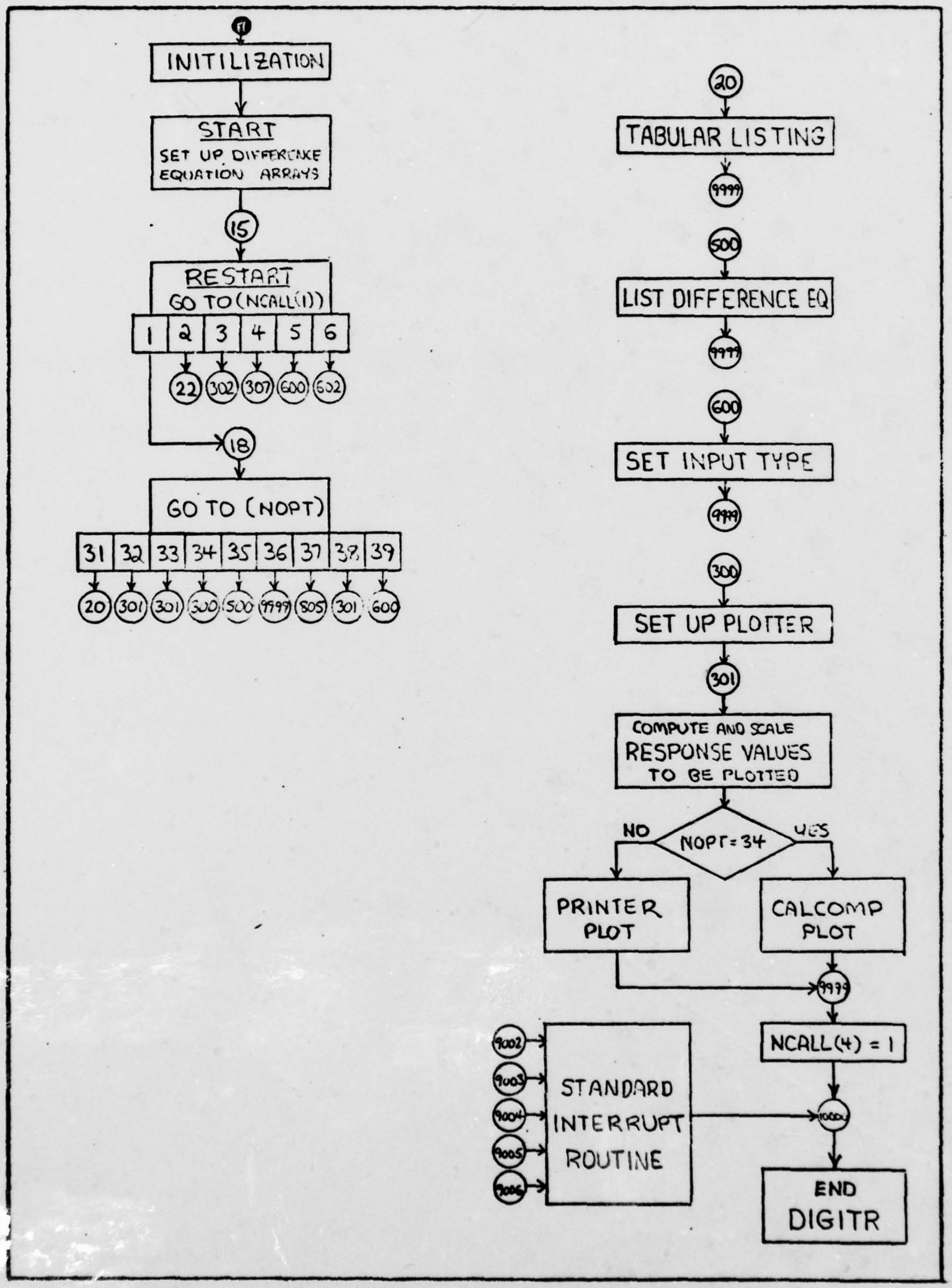


Fig. 4. Flow chart for the program DIGITR.

4.5 OVERLAY (2,3) -- PROGRAM PLOTFIN.

This program finishes up the Calcomp plots for PARTL and DIGITR. Both programs are responsible for drawing their own actual response curves, but PLOTFIN does the rest including boxes, grids, title, time axis, and tic marks. The actual coding itself is very straightforward and needs little explanation.

4.6 OVERLAY (3,0) -- PROGRAM POLY.

POLY handles all transfer function input and polynomial operations for TOTAL. It is responsible for 17 options, including options 2 through 9 and 61 through 69, making it one of the largest overlays in the program. Its structure, however, is very simple.

POLY is divided into 17 sections which are addressed by branches of computed GO TO statements. For a given option number, only one of these sections is executed before control is returned to the main overlay. Thus, each section is essentially independent of the others and it is not difficult to understand the operation of any particular option.

The understandability of POLY is further enhanced by extensive use of subroutines in its design. This modular approach allows a person to study the program "from the top down" by treating lower-level routines as "black boxes" while studying those at a current level. Subroutines used in POLY are discussed in detail in Section 5.

POLY does have one unique feature that is worthy of mention: the use of an interrupt request to factor polynomials.

Because of the program's size and the large number of subroutines it uses, there was not enough room for everything in a single overlay. Therefore, the factoring subroutine DMULR was moved to a separate overlay (overlay 10) to make room for the rest.

With this approach, when polynomial roots are needed, POLY terminates temporarily, overlay 10 is executed, and POLY is restarted, beginning execution where it left off. The polynomial to be factored and its resulting roots are passed between the two overlays with a labeled common statement.

The procedure described above is called the "standard interrupt routine." It is initiated by a series of FORTRAN statements such as those found beginning at statement 8002 in POLY:

NRPT=1	Tell the executive (overlay(0,0))that this is an interrupt request, not a normal termination.
NROUTE(1)=10	Tell the executive that this is an interrupt request for overlay 10.
NROUTE(2)=3	Tell the executive to return control to overlay 3 (POLY) when overlay 10 has finished.
NCALL(1)=12	Tell the RESTART routine at the beginning of POLY to jump to the 12th statement number in its GO TO array when POLY is restarted (so that POLY will pick up at statement 8003 where it left off).
GO TO 1000	Send control to the executive so that it can begin the above set of instructions.

The net effect of this interrupt routine is that it allows one primary overlay to call another -- a feat not normally possible with overlays. Further information on this technique is presented in the discussion of subroutine READS in Section 5.

4.7 OVERLAY (4,0) -- PROGRAM ROOT10.

ROOT10 is the executive primary overlay which controls the four secondary overlays that do all the root locus options in TOTAL. It is essentially the original AFIT program ROOTL, (Ref. 2) although it has been modified and improved considerably.

A general flow diagram for ROOT10 is given in Fig. 5. The secondary overlays it calls are described in the following sections.

4.8 OVERLAY (4,1) -- PROGRAM ROOT11.

ROOT11 sets up the Calcomp plot. It is responsible for drawing everything but the actual locus branches, including boxes, titles, axes, labels, poles, and zeros. In addition, this routine performs the search for locus points at a specified ZETA of interest. In s-plane the search is done along a constant zeta line, beginning at a distance RAD from the origin. In z-plane, the search begins at an undamped natural frequency ω_n = RAD and spirals from the z = 1 + j0 point in to the origin. ROOT11 draws the zeta line of interest and, in z-plane (TSAMP $\neq$ 0), it draws the unit circle.

4.9 OVERLAY (4,2) -- PROGRAM ROOTS.

ROOTS performs only one function. For a given value of GAIN, it calculates the closed-loop transfer function in polynomial form. It then factors the denominator polynomial and prints out the closed-loop poles at the GAIN of interest.

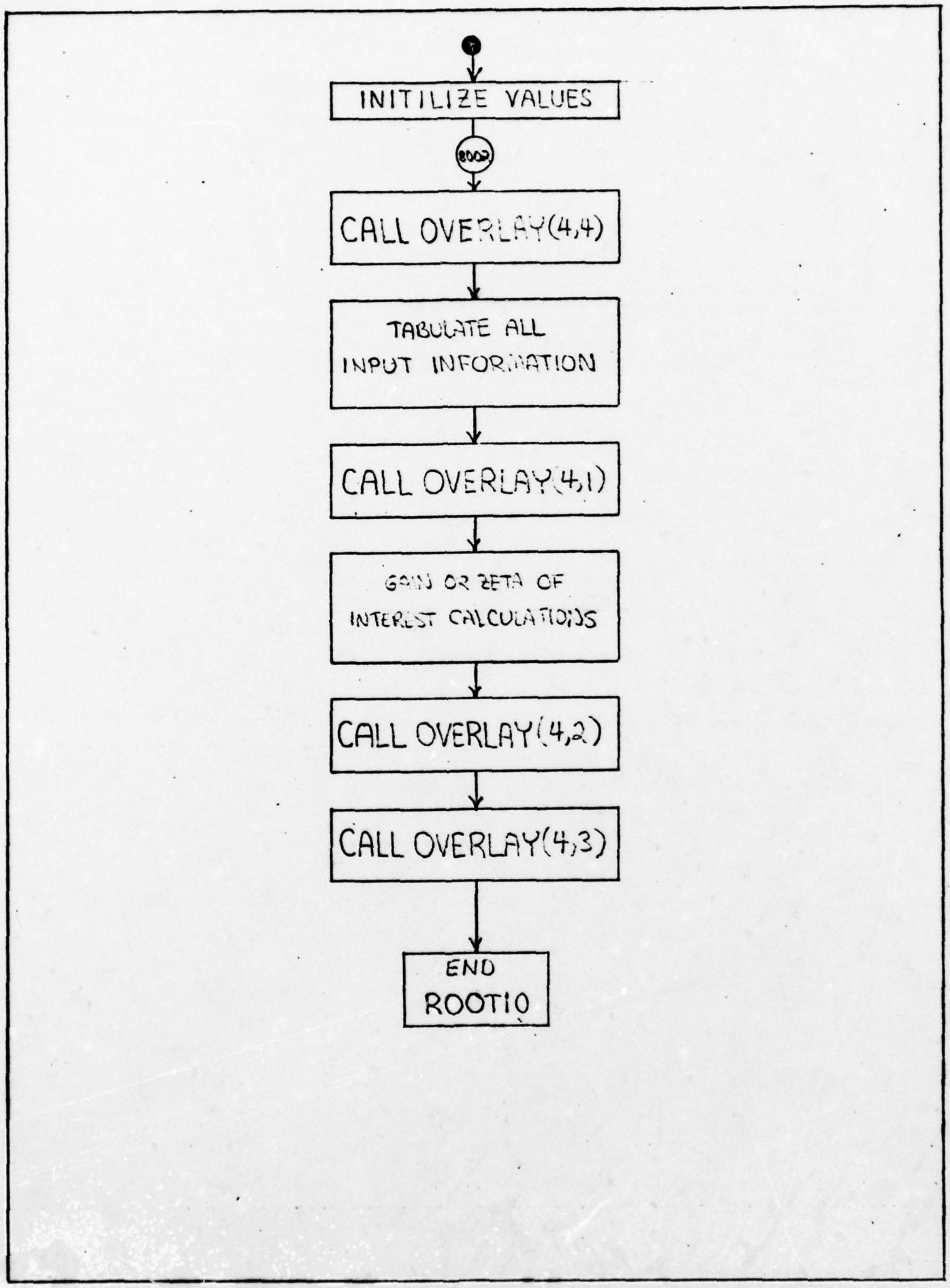


Fig. 5. Flow chart for the program ROOT10.

4.10 OVERLAY (4,3) -- PROGRAM ROOT12.

ROOT12 is the workhorse of the root locus programs. It computes the points on each branch of the locus within the boundaries of calculation.

The method used to calculate the root locus is an iterative search for one locus point at a time. Since points on the locus are only desired within a set of boundaries specified by the user, there are four types of root locus branches which may occur:

1. A branch which lies entirely within the boundaries.
2. A branch which starts inside the boundaries and goes outside.
3. A branch which starts outside and enters the region of calculation, or which reenters the region after having left it.
4. A branch which lies entirely outside the boundaries.

Branches which start inside the boundaries are found by beginning at each pole within the region and using a circular search routine (to be described later) to follow the branch until it reaches a zero or goes outside the boundary. Branches which enter the region from outside are found by searching the boundaries for a branch crossing and then following the branch with the circular search routine until it terminates at a zero or leaves the region again. Branches which lie entirely outside the region of calculation are skipped and an informative message printed.

The circular search routine mentioned above is used after the first point on a given branch has been located. This routine simply searches around a circle of radius DEL centered at the point already known until another locus point is found which satisfies the root locus 180 or 0 degree angle criterion. The new locus point is then used as the center of another circular scan and the process repeats until a point is found which lies outside the boundaries or which is less than DEL units from a zero.

Fig. 6 shows a flow chart depicting how each branch is located by ROOT12. First, the branches beginning at each pole are traced until they leave the boundaries or terminate at zeros. Then the boundaries are searched for other incoming branches and these branches are traced. When the entire boundary of the region of interest has been searched, every locus branch has been found and ROOT12 returns control to the primary overlay ROOT10.

4.11 OVERLAY (4,4) -- PROGRAM ADAPT.

This overlay was written to allow interfacing of subroutine READS with the other ROOTL overlays. Subroutine READS is used throughout TOTAL in place of the standard FORTRAN READ statement to receive data interactively while protecting against input errors. Because of its size, there was not enough space for it in any of the existing ROOTL overlays so a new overlay, ADAPT, was written.

ADAPT is simply an input program designed to ask the user for all the information needed for a given option number. It prints out all prompting messages, calls READS to receive the user's input, and stores the input in the common data base for use by the other root locus overlays.

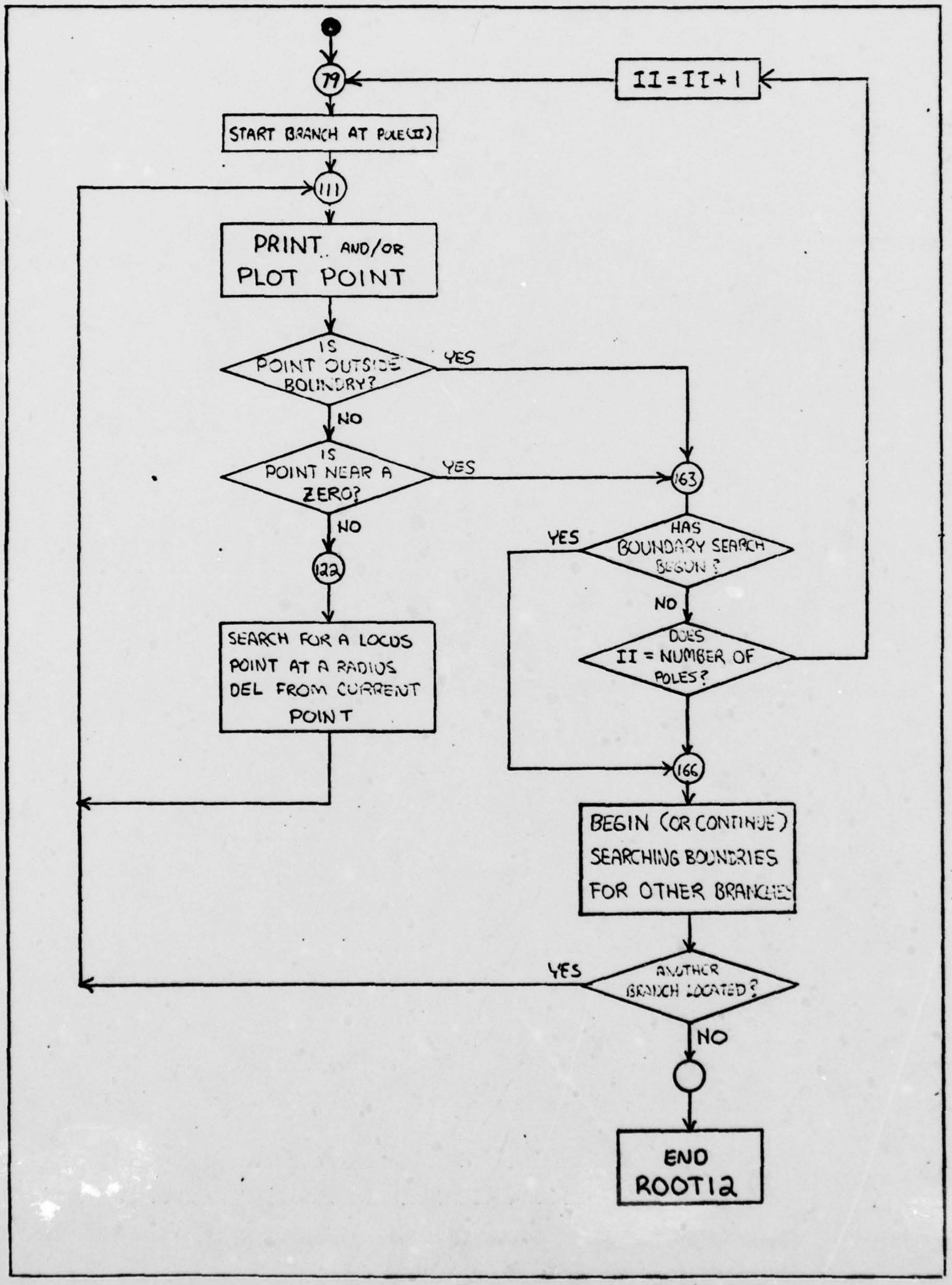

79
START BRANCH AT POLE(II)
111
PRINT AND/OR PLOT POINT
IS POINT OUTSIDE BOUNDRY?
YES
NO
IS POINT NEAR A ZERO?
YES
NO
122
SEARCH FOR A LOCUS POINT AT A RADIUS DEL FROM CURRENT POINT
II = II + 1
163
HAS BOUNDARY SEARCH BEGUN?
YES
NO
DOES II = NUMBER OF POLES?
166
BEGIN (OR CONTINUE) SEARCHING BOUNDRIES FOR OTHER BRANCHES
ANOTHER BRANCH LOCATED?
YES
NO
END ROOT12

Fig. 6. Flow chart for the program ROOT12.

4.12 OVERLAY (5,0) -- PROGRAM FREQR.

FREQR is a very short executive program which simply calls two secondary frequency response overlays. The first overlay called, program FREQOUT, performs all of the actual frequency response options. If a Calcomp plot is generated, the second overlay, program PLOTSET, is called to finish the details of the plot. FREQOUT and PLOTSET are described in Sections 4.13 and 4.14.

4.13 OVERLAY (5,1) --PROGRAM FREQOUT.

FREQOUT performs all discrete and continuous frequency response options. A general flow chart for the program is given in Fig. 7.

Execution normally begins at statement 10. After a few set-up operations, a seven branch computed GO TO statement is executed to send the program flow to the section of the program responsible for executing the given option number, NOPT. These seven sections (one for each frequency response option) are independent of each other and they all return control to statement 9999 which terminates FREQOUT.

Because FREQOUT has been divided into small, independent sections of coding, it is easy for a programmer to study any particular option of interest. The only sections which really need further explanation are the Calcomp plot options (55 and 56). These will be discussed shortly.

FREQOUT uses two function subprograms to do the actual frequency response calculations. Function FW(W) returns the magnitude of the response for a given value of frequency W.

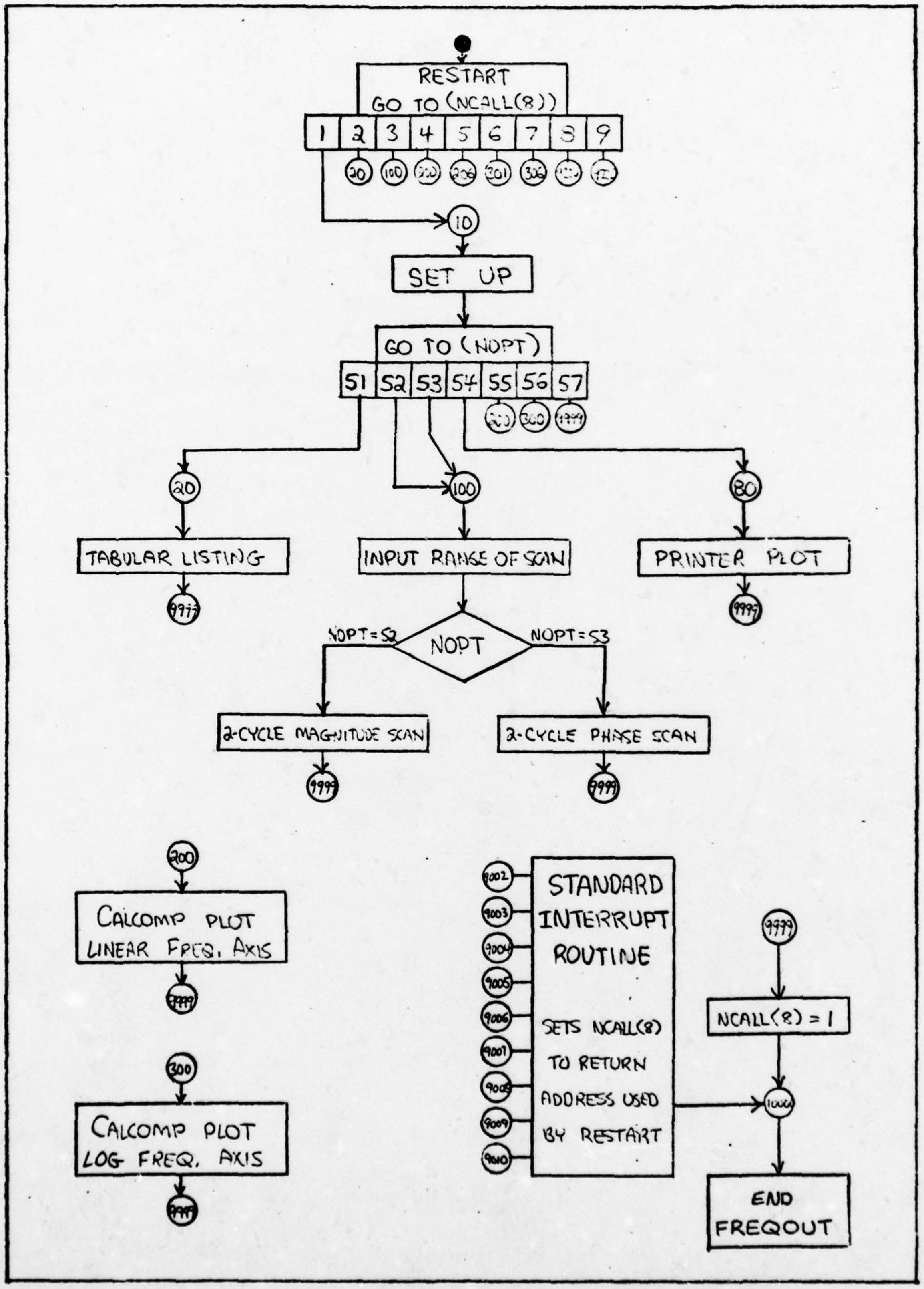


Fig. 7. Flow chart for the program FREQOUT.

Function AW(W) returns the corresponding phase angle. The units of frequency used (hertz or radians per second), the magnitude returned (linear or decibels), and the angle returned (degrees or radians) are determined internally by FW and AW based on the values of the logical variables HERTZ, DECIBEL, and DEGREE respectively. Whether the values returned are for open-loop or closed-loop transfer functions, and whether the function is treated as discrete or continuous, is also determined inside FW and AW based on values of CLOSED and TSAMP respectively. The values of these variables are specified by the user prior to selecting the option desired as described in Section 2.10 (option 93) of the User's Manual for TOTAL. FW and AW are described completely in Section 5 of this manual.

As mentioned earlier, the Calcomp plotting routines (options 55 and 56) are of sufficient complexity to merit further explanation. The expanded flow chart of these routines given in Fig. 8 should be helpful in understanding the following discussion.

Options 55 and 56 differ only in that the first uses a linear frequency axis while the second uses a logrithmic one. Because of the many similarities, much of the coding is identical and has been combined. In fact, after obtaining the magnitude and/or angle values at the appropriate frequencies, both options use the same routine to finish scaling and plotting the data.

Option 55 begins at statement 200 by requesting the user to specify a frequency range of interest. Option 56 begins

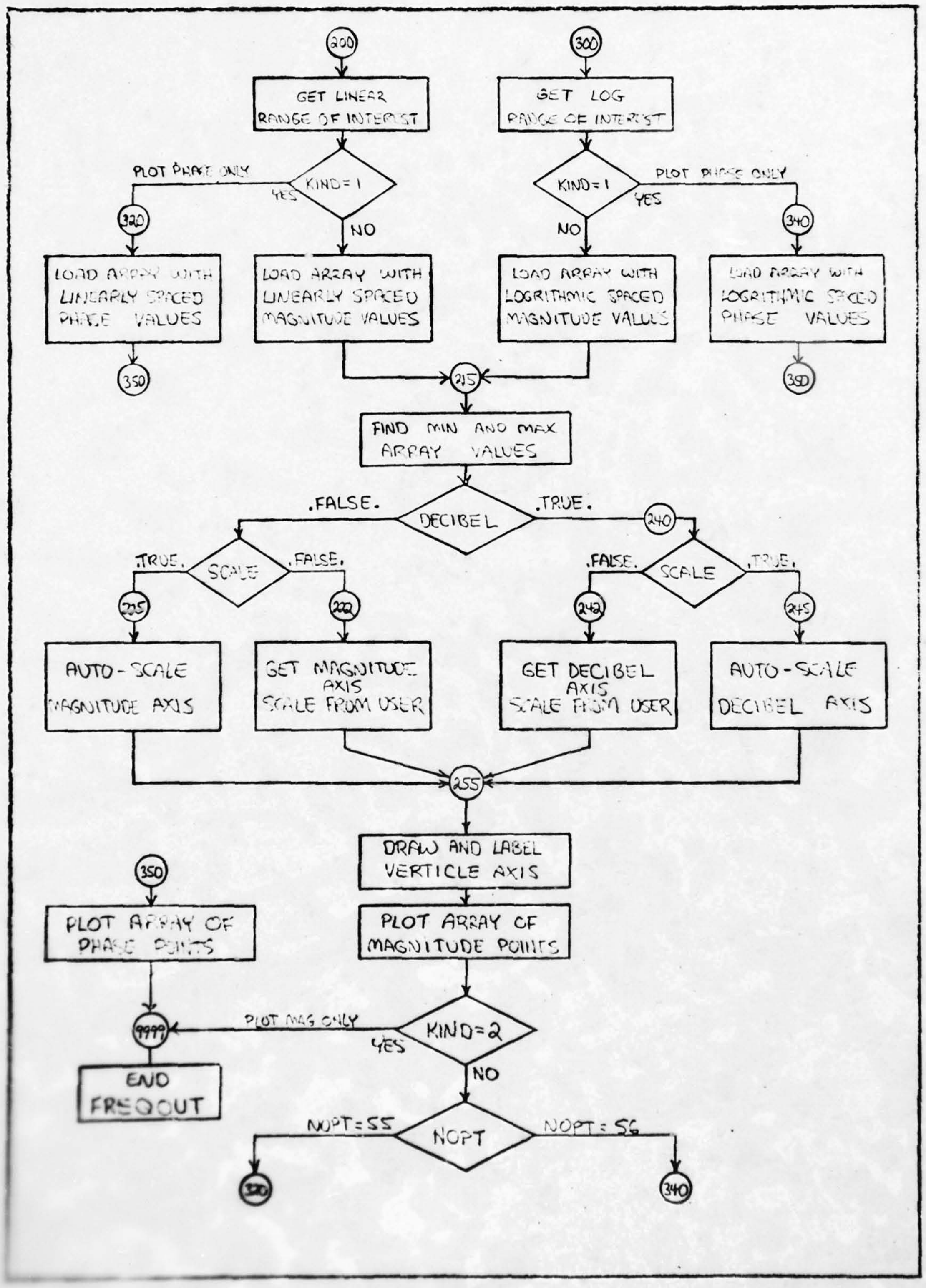

200
300
GET LINEAR RANGE OF INTEREST
GET LOG RANGE OF INTEREST
PLOT PHASE ONLY
KIND=1
YES
NO
320
340
LOAD ARRAY WITH LINEARLY SPACED PHASE VALUES
LOAD ARRAY WITH LINEARLY SPACED MAGNITUDE VALUES
LOAD ARRAY WITH LOGRITHMIC SPACED MAGNITUDE VALUES
LOAD ARRAY WITH LOGRITHMIC SPACED PHASE VALUES
350
215
FIND MIN AND MAX ARRAY VALUES
.FALSE.
DECIBEL
.TRUE.
240
SCALE
225
222
242
245
AUTO-SCALE MAGNITUDE AXIS
GET MAGNITUDE AXIS SCALE FROM USER
GET DECIBEL AXIS SCALE FROM USER
AUTO-SCALE DECIBEL AXIS
255
DRAW AND LABEL VERTICLE AXIS
PLOT ARRAY OF MAGNITUDE POINTS
PLOT ARRAY OF PHASE POINTS
9999
PLOT MAG ONLY
KIND=2
END FREQOUT
NOPT=55
NOPT
NOPT=56

Fig. 8. Flow chart for FREQOUT's Calcomp plot routines.

in a similar fashion at statement 300. Both options then ask the user to specify whether to plot phase only, magnitude only, or both. The value of the variable "KIND" is specified by the user's choice as: 1 = plot phase, 2 = plot magnitude, or 3 = plot both. KIND is then used as an internal flag to control flow throughout the rest of the routine.

The first set of calculations performed by either option is to fill the variable "ARRAY" with 800 phase or magnitude values depending whether KIND = 1 or KIND = 2. If KIND = 3, the magnitude values are computed, scaled, and plotted first and the process then repeated for the phase.

When the calculations are complete, ARRAY is just an array of 800 points to be plotted at even intervals and it is no longer necessary to know whether the points are spaced linearly or logrithmically. (Logrithmically spaced points on a log axis are drawn the same as linearly spaced points on a linear axis.) Thus, both options transfer control to statement 215 at this point.

Statement 215 begins the magnitude scaling routine for an arbitrary array of 800 magnitude points. These points are scaled either linearly or in decibels, depending on the value of the logical variable DECIBEL. Once this selection is made, the user is asked to specify the desired scale, or the array is scaled automatically, depending on the logical variable SCALE which the user can define. (See option 93 in Section 2.10 of the <u>User's</u> <u>Manual</u>.)

After the axis scale has been determined, flow goes to statement 255 and the magnitude array and axis are plotted. The program then terminates or, if KIND = 3, it cycles back around to statement 320 or 340 to load, scale and plot the phase response.

When FREQOUT finally does end option 55 or 56, it returns control to the primary overlay FREQR, which calls program PLOTSET to finish the plot.

4.14 OVERLAY (5,2) -- PROGRAM PLOTSET.

PLOTSET is a short program which draws the entire Calcomp plot for options 55 and 56, except for the actual response curves. Its job includes drawing the boxes, axes, title, labels, and grid lines using standard Calcomp routines.

4.15 OVERLAY (6,0) -- PROGRAM READER.

READER is the program in TOTAL responsible for all interactive functions. It receives all input from the user when in OPTION mode and translates it while checking for errors into a coded set of numerical commands which TOTAL can execute. READER also receives all input from the user in CALCULATOR mode and performs all calculator operations.

READER is essentially a compiler. It reads input from the user one 80-character line at a time and translates the character patterns into a sort of "machine language" of numbers which can be easily used for branching with FORTRAN IF and computed GO TO statements. The compiler is capable of recognizing a large variety of character patterns including: command names, variable names, indicies, numbers, equations, options, several special characters such as "$" and "?". It can also handle abbreviations if they are unique.

The compiled results from READER are stored in two arrays called MCOMM and DATM. Both arrays have a dimension of 100 and are indexed by a pointer MPT. MCOMM contains the type of an item such as whether it is a command or a variable, etc. DATM contains the specific item identification, such as <u>which</u> command or <u>which</u> variable. For example, if a series of letters was found to be the name of a variable, the fact that it was a variable is stored in MCOMM while its specific variable number is stored in DATM. READER recognizes six different types of input items. The code which it assigns in MCOMM to represent each item is shown below:

ITEM TYPE	MCOMM CODE	DATM CONTENTS
Command Name	1	Command Number
Variable Name	2	Variable Number
Numerical Data	3	Number's Value
Open Parenthesis	4	Number of indicies to follow
Equal Sign	5	Number of equated item to follow
Option	6	Option Number

The manner in which READER uses this code is best illustrated by an example. If the user types a line of input as follows:

```
    OPTION >    ECHO,ON    AMAT(8,5)=3.1415    26
```

READER would store the compiled results in MCOMM and DATM as shown in Fig. 9.

As each item in the input is coded, it is stored in the MCOMM and DATM array elements, currently addressed by the index pointer MPT. After each entry, MPT is incremented to point

MPT	INPUT ITEM	COMPILED OUTPUT ARRAYS MCOMM(MPT)	DATM(MPT)
1	ECHO	1 item is a command name	5 5th entry in TABLE
2	ON	1	2 2nd entry in TABLE
3	AMAT	2 item is a variable name	176 176th entry in TABLE
4	(	4 variable is subscripted	2 variable has 2 subscripts which follow
5	8	3 item is a number	8 value of 1st subscript
6	5	3	5 value of 2nd subscript
7	=	5 variable has been set equal to something	1 one value follows the = sign
8	3.1 + 15	3 item is a number	3.1415 value of the number
9	26	6 item is an option number	26 value of option number

FIGURE 9. Example of compiled output from READER.

at the next empty location in the arrays. When an entire line has been compiled, MPT is set equal to 1 to point at the first coded instruction and READER is terminated. These coded instructions are then interpreted by the main executive overlay (with the help of program DECODER (overlay (7,0))) and executed in sequence. When the list of instructions has been exhausted, READER is called again and the user is asked for another line.

READER has a vocabulary of nearly 200 words which serve as variable and command names which the user may type. This vocabulary is stored in an integer array called TABLE, which is currently dimensioned to allow up to 325 words. When the compiler routine encounters a string of letters (A through Z) that are separated from other characters in the input, it stores them as a unit into a variable called WORD. The contents of this variable are then compared with every name stored in the TABLE array until a unique match is found. The index number (position in the array TABLE) of the name which matches becomes the compiled code for that name. This code is then stored in the DATM array for future reference. If no match is found anywhere in TABLE, the characters in WORD are flagged as an input error.

New command or variable names may be added to TOTAL by simply adding the new word to one of the ten DATA statements which define the TABLE array. Actually, these DATA statements define ten smaller tables which are stacked "end-to-end" with EQUIVALENCE statements to form the one large table called TABLE. Each of these tables contains a particular name type as listed below:

TABLE NAME	TYPE OF NAME IT HOLDS
TABA	A table of up to 50 command names
TABB	A table of up to 50 real constant names
TABC	A table of up to 25 integer constant names
TABD	A table of up to 25 one dimensional array names
TABE	A table of up to 25 complex variable names
TABF	A table of up to 25 two dimensional array names
TABG	A table of up to 50 calculator key names
TABH	A table of up to 25 special abbreviations
TABI	A table of up to 25 auxiliary transfer function names
TABJ	A table of up to 25 auxiliary matrix names

New names are added by simply adding on to the end of the appropriate DATA statement. This will cause one of the 325 index numbers which is not already in use by another name to become active. Thereafter, anytime the new name is typed, its index number will be stored in the DATM array. The programmer can then use this number to perform any operation he wishes to define by testing for it anytime the DATM array is used throughout TOTAL.

Complete details on how program READER actually compiles the user's inputs are beyond the scope of this manual. The interested reader is referred to Appendix C for a more complete discussion.

4.16 OVERLAY(7,0) -- PROGRAM DECODER.

DECODER is responsible for decoding the MCOMM and DATM arrays set up by READER and executing the resulting instructions. A flow chart for this program is given in Fig. 10.

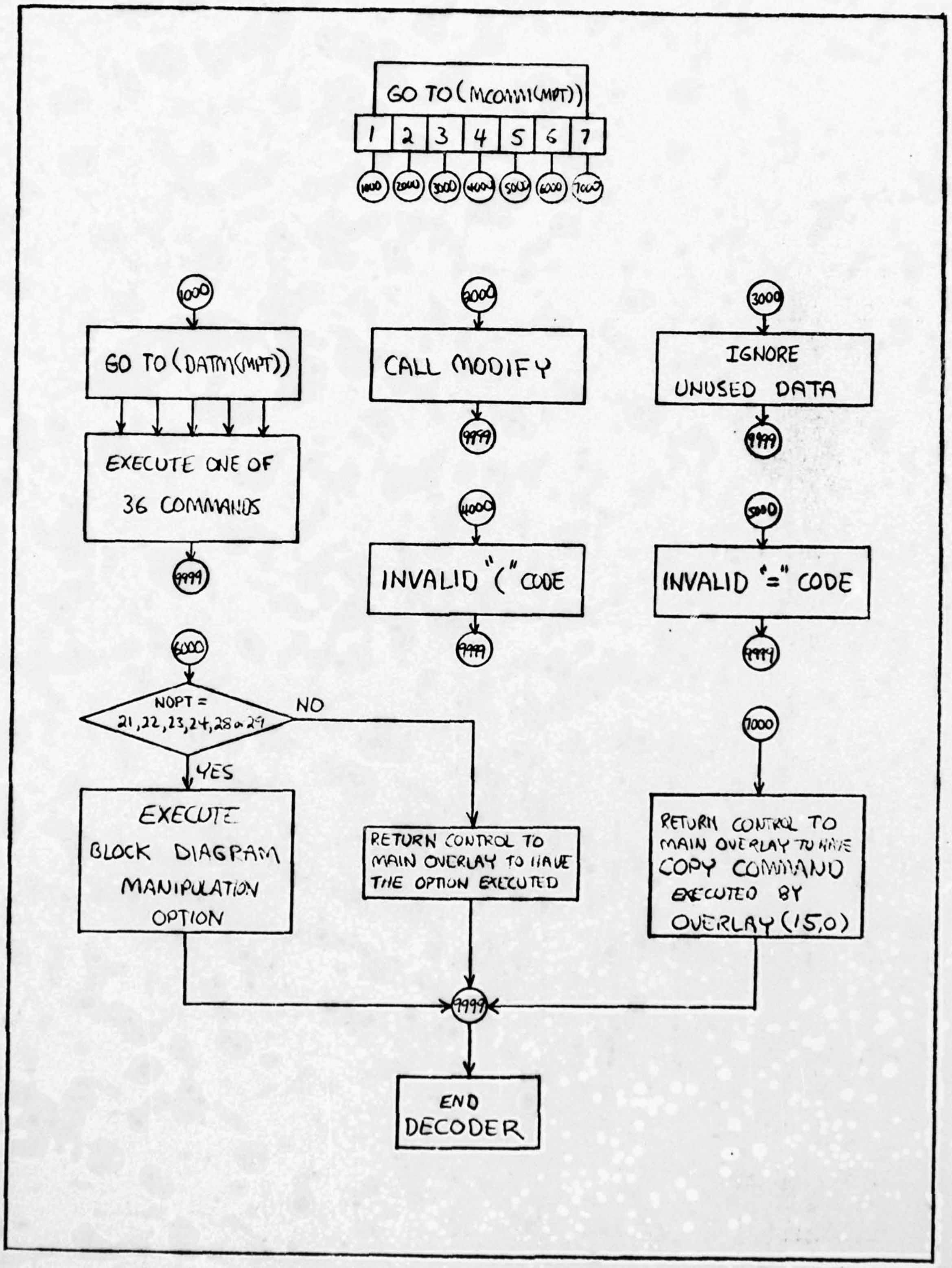

GO TO (MCOMM(MPT))
1
2
3
4
5
6
7
1000
2000
3000
4000
5000
6000
7000
GO TO (DATM(MPT))
EXECUTE ONE OF 36 COMMANDS
9999
CALL MODIFY
IGNORE UNUSED DATA
INVALID "(" CODE
INVALID "=" CODE
NOPT = 21,22,23,24,28 or 29
NO
YES
EXECUTE BLOCK DIAGRAM MANIPULATION OPTION
RETURN CONTROL TO MAIN OVERLAY TO HAVE THE OPTION EXECUTED
RETURN CONTROL TO MAIN OVERLAY TO HAVE COPY COMMAND EXECUTED BY OVERLAY (15,0)
END DECODER

Fig. 10. Flow chart for the program DECODER.

As described in Section 4.15, READER encodes input from the user as one of six possible values of the variable MCOMM (MPT) and stores additional information about the item in DATM(MPT). The code used by READER is repeated below for easy reference.

MCOMM(MPT)	ITEM NAME	DATM(MPT) CONTENTS
1	Command	Command Number
2	Varible	Variable Number
3	Numerical Data	Actual value of datum
4	Open Parenthesis	Number of indicies to follow
5	Equal Sign	Number of equated items to follow
6	Option	Option Number

Execution of DECODER begins with a computed GO TO statement which sends program flow to one of six sections of the program depending on the value of MCOMM(MPT). The remaining discussion is a detailed description of these six sections.

The Command section. If MCOMM(MPT)=1, the program flow jumps to statement number 1000. This statement is another computed GO TO statement with 36 branches selected by the value of DATM(MPT). The 36 subsections addressed in this manner each execute one of the 36 commands recognized by TOTAL. When a particular command has been executed, MPT is incremented and control is returned to the GO TO statement at the beginning of DECODER for the next instruction.

Not all of the 36 possible commands are actually executed by DECODER. Some of the command subsections only set up a request in the NROUTE array for some other overlay to be called and return control to the main executive overlay which executes the request. In the case of COPY (command number 18), an entire overlay is needed to handle all the possible variations of a single command. This overlay (Overlay (15,0)) is called directly by setting MCOMM(MPT)=7 and returning to the executive through statement number 7000. (When the executive sees MCOMM(MPT)=7, it automatically calls overlay (15,0) to execute the COPY command.)

The Variable section. If MCOMM(MPT)=2, the program jumps to statement number 2000 and subroutine MODIFY is called. MODIFY is an elaborate subroutine which lists or modifies the variable in TOTAL's common data base, whose variable number is stored in DATM(MPT). The exact action taken depends on the value of MCOMM(MPT).

If MCOMM(MPT) equals 1, 2, 3, or 6, the value of the variable is simply listed at the user's terminal. If the variable is an array, all elements of the array are listed. No further action is taken and control returns to the beginning of DECODER to fetch the next instruction.

If MCOMM(MPT+1)=4, the variable is known to be subscripted with the number of indicies stored in DATM(MPT+1). The actual index values are then obtained from DATM(MPT+2) and (if there are 2 indicies) DATM(MPT+3). If all of this information is followed by anything other than MCOMM(LASTMPT+1)=5, the value of the single array element with the given indicies is listed. If MCOMM(LASTMPT+1)=5 (where LASTMPT is the value of MPT which points to the location in DATM(MPT) where the last index value was obtained), the array element with the given indicies is set equal to the variable value stored in the next DATM location (DATM(LASTMPT+2)).

If MCOMM(MPT+1)=5, the variable is known to have been followed by an equal sign in the original input, and it is assigned the value of the number or variable stored in the (MPT+2) location.

When subroutine MODIFY has finished, it sets the MPT pointer to point to the next instruction following the last item of information it used in the MCOMM and DATM arrays. Control is then returned to the beginning of DECODER to get the next instruction.

Numerical data, open parenthesis, and equal sign sections. These three sections print error messages and when TOTAL is operating properly, should never be used. MCOMM(MPT)=3, 4, or 5 should only ever occur following a variable name and are intended only for use by subroutine MODIFY as described above. Since MODIFY always moves the MPT pointer past these numbers when it has finished using them, they should never be encountered when the main GO TO statement at the beginning of DECODER is executed. These sections are only included to protect against abrupt termination of TOTAL in event that an error occurs in one of the encoding or decoding routines.

Option section. If MCOMM(MPT)=6, DECODER jumps to statement number 6000.. Since the option number that must be executed may, in general be found in any one of the 17 primary overlays, DECODER simply returns control to the main executive and allows it to call the appropriate overlay needed to execute the option.

Termination of DECODER. When MCOMM(MPT)=0, the list of instructions stored in MCOMM and DATM has finally been exhausted and DECODER ends. The main executive overlay immediately calls READER and the user is asked for further instructions with the familiar prompt: OPTION > .

4.17 OVERLAY (9,0) -- PROGRAM HELP.

HELP is a program consisting predominantly of PRINT statements. Its sole purpose is to give help on each of TOTAL's options. Its structure consists of a single computed GO TO statement with a branch for each option. When a particular branch is executed, the program simply writes a short help message and then terminates, returning control to the main overlay.

4.18 OVERLAY (10,0) -- PROGRAM FRACTOR.

FRACTOR is a program whose sole purpose is to call subroutine DMULR to factor a polynomial. The program receives the polynomial to be factored from any other overlay in TOTAL through labeled common. After checking to ensure that the highest power coefficient of the polynomial is non-zero (and hence its degree correct), FRACTOR loads the polynomial into a double precision array and then calls DMULR.

When DMULR finishes factoring the polynomial, it returns the roots as real and imaginary parts in two separate double precision arrays. After zeroing the imaginary parts that are within 10^{-10} of zero, FRACTOR copies the results into an array in labeled common where they can be used by other programs and returns control to the main overlay.

4.19 OVERLAY (11,0) -- PROGRAM TTYPLOT.

TTYPLOT produces a printer plot of the root locus for option 48. It is called only after all points on the root locus have been calculated and stored as x-y coordinate pairs on local file TAPE10 by the root locus programs (overlay (4,0)). TTYPLOT then reads these points from TAPE10 and puts them on the plot.

The printer plot is first set up in an array called GRAPH(61,71) by storing alphanumeric characters for the border, axes, grid, poles, zeros, and locus points at appropriate locations in the 61 x 71 matrix. The plot is then generated by printing the GRAPH array contents, one row at a time, in alphanumeric format.

Points are placed in the GRAPH array, by mapping the region of the complex plane defined by the locus boundaries AA, BB, CC, and DD into a discrete-valued array 61 units wide and a proportional number of units long. This is accomplished by simply dividing the length of the real axis to be plotted by 60 and allowing the result to represent one unit on the real axis of the plot. Since most printers print 10 characters per inch horizontally and only 6 lines per inch vertically, one unit on the imaginary axis of the plot is chosen to be 10/6 times the size of a real axis unit. Once these units have been chosen, each locus point is <u>rounded</u> to its nearest discrete value on the plot and an alphanumeric character stored in that location in the GRAPH array.

TTYPLOT was written as a completely general program and could be used to plot any set of x-y points stored on TAPE10 (not just root locus points.)

4.20 OVERLAY (13,0) -- PROGRAM MISCELL.

This overlay is designed to serve as a catch-all for miscellaneous options which do not fit into any other overlay (either logically or in terms of actual available space). At present, MISCELL only performs one option: Option 93.

The first executable statement encountered in MISCELL is:

```
IF(NOPT.EQ.93)  GO TO 93
```

which sends program flow to statement 93 if the option number, NOPT, is 93. If NOPT $\neq$ 93, an error message is printed and MISCELL is terminated. Additional options may be added to this overlay by simply including another IF statement. For example, to add option 118, one would insert the statement: "IF(NOPT.EQ.118) GO TO 118" after the first IF statement and then add the new routine beginning statement 118.

4.21 OVERLAY (14,0) -- PROGRAM MATRIX.

MATRIX is just an executive program which directs the selection of two secondary overlays called MATS and MATOPR. These overlays are discussed in the following two sections.

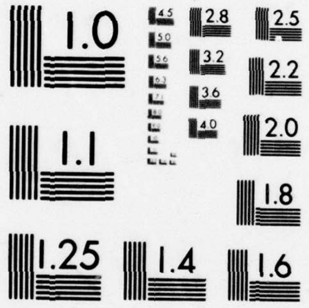
4.22 OVERLAY (14,1) -- PROGRAM MATS.

Program MATS is responsible for parts of the execution of options 11, 12, 13, 14, 15, 16, 17, 18, 19, 25, 26, 77, 78, and 79. Each of these options is performed by a separate

AD-A055 418 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
AN INTERACTIVE COMPUTER-AIDED DESIGN PROGRAM FOR DIGITAL..ETC(U)
MAR 78 S J LARIMER
UNCLASSIFIED AFIT/GGC/EE/78-2 N/L

3 OF 3
AD
A055418

END
DATE
FILMED
11-81
DTIC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

piece of coding addressed by a computed GO TO and several IF statements at the beginning of the program.

MATS is primarily designed to start every matrix-related option in TOTAL by obtaining any input which may be needed. Since options 11 through 17 involve only the input of data, MATS performs them entirely. For other options, MATS only receives the needed input and leaves the actual computations for MATOPR to perform after MATS has terminated.

By dividing the functions performed by the two programs in this manner, MATOPR does not need to use the large input subroutine READS at all and all of its space can be devoted to performing the actual matrix operations.

4.23 OVERLAY (14,2) -- PROGRAM MATOPR.

As has been mentioned, MATOPR performs most of the matrix operation options for TOTAL. Specifically, it is responsible for options 25, 26, 27, 71, 72, 73, 74, 75, and 76. MATOPR makes extensive use of subroutines in performing its many functions. These subroutines include PHOFS, FACTO, ECHOS, MADD, MATECHO, GENMMPY, MINV, TRANPOS, and GTFHTF and are discussed in Section 5.

4.24 OVERLAY (15,0) -- PROGRAM COPYIER.

COPYIER is a program designed to implement the COPY command providing the ability to transfer variables from one location to another in TOTAL's common data base. When COPYIER is called, the variable names following the COPY command are read as variable numbers from DATM(MPT+1) and DATM(MPT+2).

These variables may be of the following types:

Local polynomial arrays in labeled common
Local complex root arrays in labeled common
Local matrix arrays in labeled common
Local transfer functions in labeled common
Auxiliary matrix arrays in the mass storage file MEMAUX
Auxiliary transfer function in MEMAUX

Depending on what kind of transfer is to be performed, COPYIER selects one of ten routines to perform the operation. A list of routines which can be selected is given below:

Copy from one polynomial to another
Copy from one root array to another
Copy from one local matrix to another
Copy from a local matrix to an auxiliary matrix
Copy from an auxiliary matrix to a local matrix
Copy from one auxiliary matrix to another
Copy one local transfer function to another
Copy a local transfer function to an auxiliary one
Copy an auxiliary transfer function to a local one
Copy one auxiliary transfer function to another

Each of these operation is performed by a separate section of coding in the program. When the transfer of information is complete, COPTIER returns control to the main overlay.

4.25 OVERLAY (16,0) -- PROGRAM XFORMS.

XFORMS performs a variety of transformations between continuous and discrete transfer function representations. The program is divided into nine sections which independently perform options 81 through 89. These sections may be classified according to three types of operation as described below.

Options 81 and 84 use a partial fraction expansion technique to perform impulse invariant transformations between s and z transfer functions. In both cases, the transfer function of interest is expanded into first order terms, the individual poles transformed, one at a time, from one domain into the other, and the first order terms multiplied together to form the transformed transfer function.

Options 82, 83, 85, 86, and 89 use a single subroutine to perform substitution of the form:

s = ALPHA · (z + A)/(z + B)

or

z = ALPHA · (s + A)/(s + B)

Each option simply defines ALPHA, A, and B as needed for its particular transformation and then calls subroutine BIFORM (ALPHA,A,B) to perform the transformation on the contents of CLTF.

Options 87 and 88 simply call appropriate subroutines to perform transformations on systems represented in matrix form.

When XFORMS has completed a transformation, control is returned to the main overlay through statement number 9999.

4.26 OVERLAY (17,0) -- PROGRAM BLOCKER.

BLOCKER performs all block diagram manipulation options for TOTAL, including options 21, 22, 23, 24, 28, and 29. The program is divided into six sections addressed by a single computed GO TO statement. Each program section performs its function using a series of polynomial manipulation subroutines including POLYADD, POLYSUB, POLYMLT, FACTO, and EXPAND. These subroutines are described in Section 5. The actual operations performed using these subroutines are described in detail in Section 2.3 of the User's Manual for TOTAL.

When blocker finishes an option, it returns control to the main executive overlay through statement number 9999.

SECTION 5. DESCRIPTION OF SUBPROGRAMS

This section is written to document each of the subprograms used by TOTAL. It is intended to describe the function and use of each routine in sufficient detail to allow a programmer to make intelligent use of it whenever the need arises. A cross listing of subprograms and main programs used in TOTAL is presented in Table II for reference. From this table one may see which subprograms are used in any given program and also which programs use any given subprograms. In addition, page numbers are included so that Table II may also be used as a table of contents for the subprograms described in this section.

For each subprogram, a standard sequence of information is presented including a description, the calling sequence, any COMMON statements used, a definition of symbols, explanatory notes, and a list of subprograms used. When COMMON statements are tabulated, only the variables in each statement that are actually used by the subprogram are listed. Any variables not used (but presumably used elsewhere in TOTAL) are represented by asterisks to preserve variable position in the COMMON statement.

5.1 FUNCTION FT.

The function FT obtains the value of a continuous time response function for a given value of T using the function FTT and performs a superposition of two responses in the case of a pulse input.

Calling Sequence.

X = FT(T)

COMMON statements used.

```
COMMON/PARTL5/*,RWIDTH,*,*,INPUTR
```

Definition of symbols.

T = Time at which response is to be evaluated
INPUTR = 4 input is a pulse
$\neq$ 4 input is an impulse, step, ramp, or sinusoid
RWIDTH = Width of input pulse in seconds

Nôtes.

(1) If INPUTR $\neq$ 4, FT calls FTT once and simply returns the results to the calling program. If INPUTR = 4, FTT is called twice for values of time T and T-RWIDTH and the results subtracted using superstition to obtain the pulse response.

Subprograms used.

FTT

5.2 FUNCTION FTT.

The function FTT calculates the value of a continuous time response function for a given value of T using information placed in common by the program TIMER.

Calling sequence.

X = FTT(T)

COMMON statements used.

```
COMMON/PARTL1/ZZZ(50),XR,NXX,YYY(50),W(50)
COMMON/PARTL2/CR(50),EC(50),OM(50),FE(50),LL
```

Definition of symbols.

T = Time in seconds at which function is to be evaluated. The variables in COMMON form the coefficients of the function to be evaluated which has the form.

$$\text{FTT} = \sum_{I=1}^{12} \text{F1(I)} + \sum_{I=1}^{LL} (\text{F2(I)} + \text{F3(I)}) \text{ where}$$

F1(I) = ZZZ(I) * EXP(XR*T)
F2(I) = YYY(I) * EXP(W(I)*T)
F3(I) = CR(I) * EXP(EC(I)*T) * SIN(OM(I)*T + FE(I)

Preceding page blank

5.3 SUBROUTINE PROPGAT.

The subroutine PROPGAT iterates a recursive difference equation of the form:

$$c(k) = A(1)r(k) + A(2)r(k - 1) + \ldots + A(N)r(k - N + 1) - B(2)c(k - 1) - \ldots - B(N)c(k - N + 1)$$

to obtain the current value of c(k) given the past N inputs and outputs.

Calling sequence.

CALL PROPGAT (A,B,R,C,N,K)

Definition of symbols.

A = Vector array of coefficients of R terms
B = Vector array of ceofficients of C terms
R = Vector array of past N inputs (including present)
C = Vector array of past N outputs (including present)
N = Order of difference equation
K = Current index value

Notes.

(1) The value of r(k - M) is stored in R(M + 1) and similarly, the value of c(k - M) is stored in C(M + 1), and so on.

Subprograms used.

RIN

5.4 FUNCTION RIN.

The function RIN computes the current input r(k) for use by PROPGAT. It is capable of generating an impulse, step, ramp, pulse, or sinusoid depending on the value of INPUTR.

Calling sequence.

X = RIN(K)

COMMON statements used.

```
COMMON/DIGIT/TSAMP
COMMON/PARTL5/RSLOPE,RWIDTH,RHIGHT,ROMEGA,INPUTR
```

Definition of symbols.

K = Index value of the sampling instant in question
TSAMP = Sampling time in seconds

RSLOPE = Ramp input slope in units/sample
RWIDTH = Pulse input width in samples
RHIGHT = Magnitude scale factor for any input
ROMEGA = Frequency of sinusoidal input
INPUTR = 1 RIN = RHIGHT if K = 0
       = 2 RIN = RHIGHT if K $\geq$ 0
       = 3 RIN = SLOPE * K if K $\geq$ 0
       = 4 RIN = RHIGHT if 0 $\leq$ K $\leq$ RHWIDTH
       = 5 RIN = RHIGHT * SIN(ROMEGA*TSAMP*K)

5.5 SUBROUTINE POLAR.

The subroutine POLAR converts a set of cartesian coordinates AC and BD to polar form as a magnitude FACT and an angle FACTR.

Calling sequence.

CALL POLAR

COMMON statements used.

COMMON/POLARC/AC,BD,FACT,FACTR

Definition of symbols.

AC = x-coordinate
BD = y-coordinate
FACT = Magnitude = SQRT(AC**2 + BD**2)
FACTR = Angle in degrees = ATAN(BD/AC)

Notes.

(1) The angle FACTR is returned as a value between $\pm\pi$.

5.6 SUBROUTINE SPECS.

The subroutine SPECS finds the continuous time response figures of merit rise time, duplication time, peak time, setting time, peak value, and final value. It also writes these values to an output device.

Calling sequence.

CALL SPECS

COMMON statements used.

COMMON/TOTL14/*,*,*,*,*,*,*,*,N60
COMMON/PARTL1/ZZZ(50),XR,NXX,YYY(50),W(50)

```
COMMON/PARTL2/CR(50),EC(59),OM(50),FE(50),LL
COMMON/PARTL3/FINVAL,DEL
```

Definition of symbols.

```
NGO = 6  Output is written to file ANSWER
    = 7  Output is written to user's terminal
FINVAL = Final value of response computed by program TIMER
DEL = Iteration step size
```

The remaining variables listed above are defined in Section 5.2.

Notes.

(1) This subroutine finds every peak in the time response over a range of time from 0 to eight times the longest time constant, and picks the biggest one.

Subprograms used.

FT   ZEROIN   PEAK

5.7 SUBROUTINE ZEROIN.

The subroutine ZEROIN finds the first value of T after T = TMIN, where the function FT(T) equals some specified value GOAL using an iterative search procedure..

Calling sequence.

```
CALL ZEROIN(T,GOAL,TMIN,TMAX,LUCK)
```

COMMON statements used.

```
COMMON/PARTL3/FINVAL,DEL
```

Definition of symbols.

```
T = Value of time where FT(T) = GOAL
GOAL = Value of FT for which a corresponding time is to
       be found
TMIN = Starting value of time range to be searched
TMAX = Final value of time range to be searched
LUCK = 0  No such time was found between TMIN and TMAX
     = 1  The search for T was successful
FINVAL = Final value of the function
DEL = Starting step size of search (chosen by SPECS to be
      one tenth of the shortest period of oscillation in
      the function.)
```

Notes.

(1) If the correct T cannot be found, the value of T returned is T = 0.

(2) The iterative search used is continued until FT(T) is within 10^{-6} * FINVAL of the specified GOAL.

Subprograms used.

FT

5.8 SUBROUTINE PEAK.

The subroutine PEAK finds the first value of T after T = TMIN where the slope of FT(T) is zero, using an iterative search technique.

Calling sequence.

CALL PEAK(T,TMIN,TMAX,LUCK)

COMMON statements used.

COMMON/PARTL3/FINVAL,DEL

Definition of symbols.

```
T = Value of time where slope of FT is zero
TMIN = Starting value of time range to be searched
TMAX = Final value of time range to be searched
LUCK = 0  No peak was found between TMIN and TMAX
     = 1  The search was successful
FINVAL = Final value of the function
DEL = Starting step size of the search
```

Notes.

(1) If the function is increasing at TMIN, the search looks for a local maximum. If the function is decreasing, the search looks of a local minimum.

(2) The search is terminated when the value of TMAX is reached or when three values of time are found close enough together that the value on either side of the peak changes by less than 10^{-6} * FINVAL units.

Subprograms used.

FT

5.9 SUBROUTINE ECHOS.

The subroutine ECHOS is an output routine which is used to print polynomial coefficients and roots is a compact table.

Calling sequence.

```
CALL ECHOS(ROOT, POLY, NP, PK)
```

COMMON statements used.

```
COMMON/TOTL14/ *, *, *, *, *, *, *, *, NGO
```

Definition of symbols.

```
ROOT(50,2)= Array of roots to be printed
POLY(51) = Array of coefficients to be printed
NP = Order of polynomial to be printed
PK = Polynomial constant to be printed
NGO = 6  Output is printed on TAPE6 = ANSWER
    = 7  Output is printed on TAPE7 = OUTPUT
```

5.10 SUBROUTINE POLECHO.

The subroutine POLECHO is an output routine which tabulates polynomial coefficients to ten decimal places with corresponding index numbers.

Calling sequence.

```
CALL POLECHO(POLY, NP)
```

COMMON statements used.

```
COMMON/TOTL14/*, *, *, *, *, *, *, *, NGO
```

Definition of symbols.

```
POLY(51) = Array of polynomial coefficients to be printed
NP = Order of polynomial
NGO = 6  Output is printed on TAPE6 = ANSWER
    = 7  Output is printed on TAPE7 = OUTPUT
```

5.11 SUBROUTINE RTECHO.

The subroutine RTECHO is an output routine designed to print out the real and imaginary parts of an array of polynomial roots to ten decimal places. The routine also prints an index number for each root.

Calling sequence.

```
CALL RTECHO(ROOT,NP)
```

COMMON statements used.

```
COMMON/TOTL14/*, *, *, *, *, *, *, *, NGO
```

Definition of symbols.

ROOT(50,2) = Array of 50 complex numbers where ROOT(I,1) is the real part and ROOT(I,2) is the imaginary part of the I th root to be printed
NP = Number of roots to be printed
NGO = 6 Output is printed on TAPE6 = ANSWER
 = 7 Output is printed on TAPE7 = OUTPUT

5.12 SUBROUTINE MATECHO.

The subroutine MATECHO is an output routine designed to print the elements of a matrix of arbitray dimensions.

Calling sequence.

```
CALL MATECHO(AMAT,NA,MA)
```

COMMON statements used.

```
COMMON/TOTL14/*, *, *, *, *, *, *, *,NGO
```

Definition of symbols.

AMAT(10,10) = Matrix of elements to be printed
NA = Number of rows in AMAT
MA = Number of columns in AMAT
NGO = 6 Output is printed on TAPE6 = ANSWER
 = 7 Output is printed on TAPE7 = OUTPUT

Notes.

(1) If the matrix to be printed has more than five columns, and NGO = 6, each row of the first five columns is tabulated first followed by any remaining columns lower on the output page. If NGO = 7, all columns are tabulated side-by-side.

5.13 SUBROUTINE TFECHO.

The subroutine TFECHO is an output routine which tabulates the numerator and denominator polynomial coefficients and roots of a transfer function in a compact form.

Calling sequence.

CALL TFECHO(NAME,LET,ZERO,POLE,POLYD,GNP,GDK,NZ,NP)

COMMON statements used.

COMMON/TOTL14/*, *, *, *, *, *, *, *, NGO

Definition of symbols.

NAME = Left justified Holerith constant containing the initials of the transfer function to be printed. These initials have the letters TF appended to them forming the transfer function name.
LET = Number of letters stored in NAME (usually 1 or 2)
ZERO(50,2) = Complex array of transfer function zeros
POLE(50,2) = Complex array of transfer function poles
POLYN(51) = Array of denominator coefficients
GNP = POLYN(1)
GDP = POLYD(1)
NZ = Order of transfer function numerator
NP = Order to transfer function denominator
NGO = 6 Output is printed on TAPE6 = ANSWER
= 7 Output is printed on TAPE7 = OUTPUT

5.14 SUBROUTINE POLYADD.

The subroutine POLYADD adds polynomial A and polynomial B to form polynomial C.

Calling sequence.

CALL POLYADD(A, B, C, NA, NB, NC, AK, BK,CK)

Definition of symbols.

A(51),B(51) Polynomials to be added where A(1) is the highest order coefficient of polynomial A and so on.
C(51) = Resulting sum of A and B
NA,NB,NC = Order of A, B, and C respectively
AK = A(1)
BK = B(1)
CK = C(1)

5.15 SUBROUTINE POLYSUB.

The subroutine POLYSUB subtracts polynomial B from polynomial A to form polynomial C.

Calling sequence.

```
CALL POLYSUB(A,B,C,NA,NB,NC,AK,BK,CK)
```

Definition of symbols.

See section 5.14

Subprograms used.

POLYADD

5.16 SUBROUTINE POLYMLT.

The subroutine POLYMLT multiplies polynomial A by polynomial B to form polynomial C. If the order of C is greater than 50 the routine aborts.

Calling sequence.

```
CALL POLYMLT(A,B,C,NA,NB,NC,AK,BK,CK), RETURNS (number)
```

Definition of symbols.

```
Number = Statement number in calling program to which
         control is to be returned if the product C
         has an order greater than 50.
(All other symbols -- see Section 5.14)
```

5.17 SUBROUTINE EXPAND.

The subroutine EXPAND expands the roots of a polynomial into a corresponding set of polynomial coefficients.

Calling sequence.

```
CALL EXPAND(ROOTR,ROOTI,GAIN,NF,POLY)
```

Definition of symbols.

```
ROOTR(50) = Array of root real parts
ROOTI(50) = Array of root imaginary parts
```

GAIN = Numerical constant to be multiplied together
       with the roots to form the coefficients
NF = Number of roots to be expanded
POLY(51) = Resulting array of coefficients where POLY(1)
           is the highest order coefficient and is always
           equal to GAIN.

5.18 SUBROUTINE FACTO.

The subroutine FACTO is a setup subroutine which calls subroutine DMULR to factor a polynomial.

Calling sequence.

CALL FACTO(POLYQ,ROOTQ,NQ,PQK)

Definition of symbols.

POLYQ(51) = Array of coefficients of polynomial to be
            factored where POLYQ(1) is the highest order
            coefficient.
ROOTQ(50,2) = Array of resulting roots where ROOTQ(I,1)
              is the real part of the I th root and ROOTQ(I,2)
              is the imaginary part.
NQ = Polynomial order
PQK = POLYQ(1)

Subprograms used.

DMULR

5.19 SUBROUTINE DMULR.

The subroutine DMULR is a polynomial factoring subroutine.

Calling sequence.

CALL DMULR(COE,N1,ROOTR,ROOTI)

Definition of symbols.

COE(51) = Double precision array of coefficients of the
          polynomial to be factored
N1 = Order of polynomial
ROOTR(50) = Double precision array of real parts of
            resulting factors
ROOTI(50) = Double precision array of corresponding
            imaginary parts.

5.20 SUBROUTINE MADD.

The subroutine MADD adds or subtracts matrix A and matrix B to form matrix C. If A and B do not have some dimensions, the routine aborts.

Calling sequence.

```
CALL MADD(A,B,C,NA,NB,NC,MA,MB,MC,S),RETURNS(number)
```

Definition of symbols.

A(10,10),B(10,10)	Input matrices
C(10,10)	Output matrix
NA,NB,NC	Number of rows in A, B, and C
MA,MB,MC	Number of columns in A, B, and C

S = +1 B is added to A
  = -1 B is subtracted from A
Number = The statement number in the calling program to which control is to be returned if the subroutine aborts.

Notes.

(1) Every coefficient in the B matrix is multiplied by S and the result added to the A matrix.

5.21 SUBROUTINE GENMMPY.

The subroutine GEN,,PY post-multiplies the matrix A by the matrix B and stores the results in matrix C. If A and B do no conform, the routine aborts.

Calling sequence

```
CALL GENMMPY(A,B,C,NA,NB,NC,MA,MB,MC),RETURNS(number)
```

Definition of symbols.

(See Section 5.20)

5.22 SUBROUTINE MINV.

The subroutine MINV performs the inversion of a square, non-singular matrix of maximum size 10 x 10. If the matrix is not square, singular, or too big, the routine is aborted.

Calling sequence.

CALL MINV(A,B,N,M),RETURNS(number)

Definition of symbols.

A(10,10) = Input matrix
B(10,10) = Inverted output matrix
N = Number of rows in A and B
M = Number of columns in A and B
Number = The statement number in the calling program to which control should be returned if the routine aborts.

5.23 SUBROUTINE TRANPOS.

The subroutine TRANPOS transposes (exchanges rows and columns) the matrix A to form C.

Calling sequence.

CALL TRANPOS(A,C,NA,MA,NC,MC)

Definition of symbols.

A(10,10) = Input matrix of maximum size 10 x 10
C(10,10) = Transposed output matrix
NA,NC = Number of rows in A and C
MA,MC = Number of columns in A and C

5.24 SUBROUTINE MATIN.

The subroutine MATIN is an input routine which interactively requests the input of a matrix one row (or column) at a time.

Calling sequence.

CALL MATIN(AMAT,NA,MA,NAME),RETURNS(input)

Definition of symbols.

AMAT(10,10) = Matrix to be input
NA = Number of rows in AMAT
MA = Number of columns in AMAT
NAMS = Alphanumeric name of matrix to be input
Input = Statement number in calling program to which control is to be returned if the subroutine READS requests an interrupt to call calculator or HELP.

Subprograms used.

READS MATECHO

5.25 SUBROUTINE PHOFS.

The subroutine PHOFS uses Leverrier's Algorithm to compute adj(sI - a) and det(sI - A). If the input A matrix is not square the routine is aborted.

Calling sequence.

CALL PHOFS(A,NA,MA,DET),RETURNS(number)

COMMON statements used.

COMMON/ADJNT/ADJ(10,10,10)

Definition of symbols.

A(10,10)	= A square input matrix
NA	= Number of rows in A
MA	= Number of columns in A
DET(11)	= Array of coefficients of the polynomial det(sI - A) where DET(1) is the lowest order coefficient
ADJ(10,10,10)	= Adjoint matrix adj(sI - A) where the first two indices define the row and column number of each matrix element and the third is an index on the polynomial coefficients of each element
Number	= Statement number in calling program to which contol is to be returned if the routine is aborted.

Subprograms used.

MMPY

5.26 SUBROUTINE MMPY.

The subroutine MMPY is a special matrix multiply routine used by PHOFS and EXPAND. It multiplies matrices A and B and stored the product in C.

Calling sequence.

CALL MMPY(A,B,C,M,K,N)

Definition of symbols.

A(M,K),B(K,N)	Input matrices
C(M,N)	Output Product Matrix
M,K,N	Matrix dimensions

5.27 SUBROUTINE CADJB.

The subroutine CADJB computes the polynomial G(s) from the matrix product $(C)^T$(adj(sI - A))B, where the adjoint matrix is supplied as ADJ(10,10,10).

Calling sequence.

CALL CADJB(C,B,G)

COMMON statements used.

COMMON/ADJ(10,10,10)

Definition of symbols.

C(10) = Input $\hat{c}$ vector
B(10) = Input b vector
ADJ(10,10,10) = Input matrix containing adj(sI - A)
G(10) = Output array of polynomial coefficients of the form:
$G(1)s^9 + G(2)s^8 + G(3)s^7 + \ldots + G(8)s^2 + G(9)s + G(10)$

Notes.

(1) The subroutine PHOFS (or its equivlent) must be called prior to calling CADJB to define the values of ADJ(10,10,10).

5.28 FUNCTION FW.

The function FW caluclates the discrete or continuous open or closed-loop frequency response magnitude in linear magnitude or decibels for a given frequency in hertz or radians per second.

Calling sequence.

F = FW(W)

COMMON statements used.

```
COMMON/TOTL11/OLNPOLY(51), OLDPOLY(51), OLZERO(50,2),
OLPOLE(50,2)
+NOLZ,NOLP,OLK,OLNK,OLDK
COMMON/TOTL12/CLNPOLY(51), CLDPOLY(51), CLZERO(50,2)
CLPOLE(50,2)
+NCLZ,NCLP,CLK,CLNK,CLDK
COMMON/DIGIT/TSAMP
COMMON/LOGIC2/CLOSED, DECIBEL, HERTZ, DEGREE
```

Definition of symbols.

NOLZ = Degree of OLNPOLY polynomial
NOLP = Degree of OLDPOLY polynomial

NCLZ = Degree of CLNPOLY polynomial
NCLP = Degree of CLDPOLY polynomial
W = Frequency at which the response magnitude is to be evaluated. If HERTZ = .TRUE., this frequency is assumed to be in hertz. If not, the value of W is assumed to be in rad/sec.
All other variables are defined in Section 4 of the User's Manual for TOTAL.

5.29 FUNCTION AW.

The function AW(W) is identical to FW (see Section 5.28) except that the value returned is the phase angle of the discrete or continuous open or closed-loop transfer function frequency response.

5.30 SUBROUTINE READS.

The subroutine READS is an elaborate interactive input routine which provides complete error protection and recovery and allows the user to retain control of the calling program even when it is awaiting some numerical data input. It is designed to be used in place of the standard FORTRAN READ statement whenever any input is needed.

Calling sequence.

```
CALL READS(DATN,NO), RETURNS(INRUPT,REPEAT)
```

COMMON statements used.

```
COMMON/TOTL16/X,Y,Z,T REG(20)
COMMON/TOTL17/MCOMM(100),DATM(100),MPT
COMMON/TOTL18/NRPT,NROUTE(10),NPT
```

Definition of symbols.

DATN(60) = Array in which input numbers received are returned to the calling program.
NO = Number of input numbers that READS is supposed to obtain before returning control to the calling program.
INRUPT = Actual numerical statement number in the calling program to which control is to be returned if the user requests a temporary interrupt (or complete abort) from entering data by typing a "?", "C", or "$". (See Note (1) below.)

REPEAT = Actual numerical statement number in the calling program of the user prompt statement which printed the request for input prior to calling READS. This feature allows READS to repeat user prompts when necessary (after any interrupt) by temporarily returning control to the calling program.

X,Y,Z,T REG(20) = Variables in COMMON from which the user can tell READS to get a requested piece of data instead of typing the number itself. (See Note (1) below.)

MCOMM(100),DATM,MPT = Optional variables which may contain a sequence of commands to be executed by the calling program where MPT is the index number of the command currently being executed. The only thing READS does with these variables is set them to zero if the user requests a complete abort with a "$". (See Note (1) below.)

NPT = Number of variables which have been currently input by the user. When NPT = NO, READS has completed its task and returns control normally to the statement immediately following the CALL READS statement in the main program.

NROUTE(10) = Array used by the "standard interrupt routine" (See Note (2)) to specify the code number for the routine to be called in the main program during a temporary interrupt request by the user.

NROUTE(1) = 8 if the user types a "C" interrupt request
= 9 if the user types a "?" interrupt request

NRPT = Index on NROUTE(NRPT) which is set equal to 1 during an interrupt request.

Notes.

(1) When input is requested by READS the user may:

Enter the requested numbers or

Type "C" to request a temporary return to the calling program at statment number INRUPT. NROUTE (1) is set to 8 to indicate that this is a "C" interrupt. This function is designed to allow the user to temporarily jump to a calculator routine from the middle of inputting data (although it can be used for any kind of interrupt desired).

Type "?" to request a tempory return to the calling program at statement number INRUPT. NROUTE(1) is set to 9 to indicate that this is a "?" interrupt. This function is designed to allow the user to temporarily jump to a help routine for explanation of the input needed (although it can be used for any kind of interrupt desired.)

Type "L" to list the current values of the requested variables. The first NO elements of the DATN array are listed.

Type "*" to leave the current value of a requested variable unchanged. NPT is incremented by one and READS waits for the next number in the sequence. (If NPT = NO, READS ends.)

Type "$" to completely abort not only the current input of data but also the current option being executed and any additional options which may have been stored in the MCOMM and DATM arrays awaiting execution.

Type "X" or "Y","Z","T", or "R1" through "R20" to tell READS to get the current number being requested from the corresponding variable in labeled COMMON X, Y, Z, T, or REG(1) through REG(20) respectively. READS stores the specified value into DATN(NPT), increments NPT, and looks for the next requested value. If NPT = NO, READS ends.

Type "P1" to "Pn" to set NPT equal to any number between 1 and n = NO. This allows the user to skip around in list of variables requested to enter only certain variables or to modify the values of variables already input. When Pn is typed, READS waits for the user to specify the value of the n th variable and then continues to increment NPT from that point.

(2) The subroutine READS initiates what is called the "standard interrupt routine" for whatever program calls it whenever a user types a "C", "?" or "$". This interrupt routine is just a sequence of commands stored in the array NROUTE which direct the flow of the calling program until the interrpution has ended. It works as follows: When READS encounters an interrupt character, it stores the code number of the program which is to be called in NROUTE(1), sets the pointer NRPT = 1 and returns control to statement number INRUPT in the calling program. This statement then stores the code number for the program which contains it in NROUTE(2) and returns control to the main executive overlay. The executive sees that NRPT = 1, so it calls whatever program has been specified in NROUTE(1) and increments NRPT. When the program called has finished executing the executive calls the program specified in NROUTE(2) which in turn calls the subroutine READS again and the interrupt procedure is over. READS then repeats the prompt to the user and all data that had been entered prior to the interrupt (as a reminder) and waits for the user to enter the remaining data (or request another interrupt).

Bibliography

1. AFIT/EN. ROOTL User's Manual. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, March 1975.

2. AFIT/EN. PARTL. Heaviside Partial Fraction Expansion and Time Response Program. (Revised) Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, March 1975.

APPENDIX C

DETAILS ON PROGRAM READER

(MARCH 1978)

Appendix C

This appendix is intended to provide supplimental information on the program READER which is discussed in Section 4.15 of Appendix B. Its purpose is to describe how READER handles a line of input data and how it interprets each character it encounters in the line.

How Input is Received

READER reads an 80 character line into an array of 80 words called IN(80) using a right justified, zero-fill hollerith (R1) format. This means that each 60 bit word in the array is filled with zeros except for the six least significant bits which contain the display code of a given character in the line. (A display code number is just an integer from 0 to 63 corresponding to one of the 64 possible characters which may be typed from the user's keyboard. For example, the letter "L" has a display code of 13 and is represented by the six binary digits: 001101.) Using this technique, a line of 80 keyboard characters is stored in an array 80 words long as a sequence of numbers between 0 and 63. READER is then able to operate on this array of numbers using a compiler code.

READER's Compiler Code

The compiler code used by READER is shown in Fig. 1. It consists simply of an array of binary data 64 words long which defines the meaning of each of the 64 keyboard characters in up to 20 different situations (modes of operation) in the program.

The code works like this: Associated with each keyboard character there is a 60 bit word of coded information stored in the array A. If the binary number represented by these 60 bits is written in octal (base 8), the resulting form is a 20 digit string of digits between 0 and 7. If each of these digits is considered as a piece of information (i.e. a number between 0 and 7), then each word in the array A can store 20 pieces of information about its corresponding keyboard character. This information can therefore be used to tell the compiler how to treat a particular character in 20 different situations.

How the Code is Used

At this point it may be helpful to give the reader an example of how a keyboard character can have different meanings in different situations. Suppose, for example, that the compiler is operating on a sting of keyboard characters which are intended to represent a floating point number. If any character other than 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, or "." is encountered, the compiler may assume that the current

L	A(L)	KEY	L	A(L)	KEY
1	11111111141114444166	:	33	22222222222222222222	5
2	11111111141114444111	A	34	22222222222222222222	6
3	11111111141114444111	B	35	22222222222222222222	7
4	11111111141114444111	C	36	22222222222222222222	8
5	11111111141114444111	D	37	22222222222222222222	9
6	11111111141114433111	E	38	11111111141114244132	+
7	11111111141114444111	F	39	11111111141114244132	-
8	11111111141114444111	G	40	11111111141114444136	*
9	11111111141114444111	H	41	11111111141114444136	/
10	11111111141114444111	I	42	11111111141114444166	(
11	11111111141114444111	J	43	11111111111114444166	)
12	11111111141114444111	K	44	11111111141114444144	$
13	11111111141114444111	L	45	11111111141114444166	=
14	11111111141114444111	M	46	11111111131115555166	
15	11111111141114444111	N	47	11111111131115555166	,
16	11111111141114444111	O	48	11111111141114441122	.
17	11111111141114444111	P	49	11111111141114444133	#
18	11111111141114444111	Q	50	11111111141114444166	[
19	11111111141114444111	R	51	11111111141114444166	]
20	11111111141114444111	S	52	11111111141114444166	%
21	11111111141114444111	T	53	11111111141114444166	"
22	11111111141114444111	U	54	11111111141114444166	_
23	11111111141114444111	V	55	11111111141114444166	!
24	11111111141114444111	W	56	11111111141114444166	&
25	11111111141114444111	X	57	11111111141114444166	'
26	11111111141114444111	Y	58	11111111141114444144	?
27	11111111141114444111	Z	59	11111111141114444166	<
28	22222222222222222222	0	60	11111111141114444166	>
29	22222222222222222222	1	61	11111111141114444166	@
30	22222222222222222222	2	62	11111111141114444155	\
31	22222222222222222222	3	63	11111111141114444155	^
32	22222222222222222222	4	64	11111111141115555166	;
			65	77777777757776666777	

Fig. 1 READER's Compiler Code

number has terminated and subsequent characters belong to some other input item. As long as each new character examined is 0 through 9, this condition will hold true. However, after a decimal point has been encountered, further decimal points are not allowed and the meaning of the character "." must be changed from "decimal point" to "error -- too many decimal points."

To implement the above simple example, two different meanings must be defined for the character ".". These meanings can be coded as two different numbers in two of the 20 possible entries in the element of the A array which corresponds to the character ".".

What this means is that the 64 keys can be divided into up to seven groups by each of the 20 columns of numbers stored in the A array. When a particular column is selected, the value of the number in that column corresponding to a particular character is the group to which that character belongs in the column. For example, in Fig. 1, the number 3 appears in the second column of a(L) (from the right) across from the keyboard characters: +, -, *, /, and #. Thus, these five characters belong to the same group when column two is selected. In the same column, the number 2 appears across from the keys 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and "." placing them in another group.

Continuing with the example of reading a floating point number, it is desired to divide the possible characters into groups of valid number characters, valid number terminating characters, and illegal characters to be flagged as input errors. The fourth column of the A array makes such a division, as show below:

PARTITION PERFORMED BY COLUMN 4

Number in column 4	Characters assigned to that number	Meaning of the group of characters
1	decimal point	Legal decimal point
2	0 1 2 3 4 5 6 7 8 9	Legal digit
3	E	Exponent of number
4	Letters and special characters not in any other group	Illegal characters
5	blank, comma, semicolon	Valid number terminates

When the first character of a number is encountered in the input string, the compiler goes to "mode 4" and reads subsequent characters using the partition of column 4. If the next character encountered is in group 2, it is accepted as a legal digit, the compiler remains in mode 4, and moves on to the next character. If a character in group 4 is found, it is not accepted and an error message is given. If the character belongs to group 1 (in this case it is a decimal point(the point is accepted as a legal part of the number, but the compiler shifts into mode 5 before continuing to the next character. Mode 5 is just another partition of characters, this time as defined by column 5 of the A array. The new partition is identical to the old except that the character "." has been moved from group 1 (where it meant "decimal point") to group 4 (where it means "illegal character"). For the remainder of the current number, the compiler never returns to mode 4 and a second decimal point remains illegal. Similar modes are defined by other columns to allow the characters "+" and "-" at the beginning of a number or just following the letter "E" in the exponent but no where else, and so on.

Numbers are obtained from a particular column by masking out everything except the six bits in the column of interest. These bits are then converted to their decimal equivalent and stored in the variable LGO for for use in a computed GO TO statement. Statements 205 and 260 in READER perform such operations.

VITA

Stanley J. Larimer was born on 25 June 1954 in Brookville, Pennsylvania, the son of Harold G. Larimer and Dana J. Larimer. He graduated from Brookville Area High School as valedictorian of the class of 1972 and began work on a Bachelor of Science degree in Electrical Engineering at Grove City College, Pennsylvania. In 1976 he graduated Magna Cum Laude with highest honors in electrical engineering and was commissioned Second Lieutenant as a Distinguished Military Graduate of the Air Force ROTC program. After beginning graduate work at the Pennsylvania State University under the Arnold Air Society/Link Foundation Fellowship, he entered active duty at the Air Force Institute of Technology in August 1976. He is married to the former Pamela J. Small of McMurray, Pennsylvania.

D-1